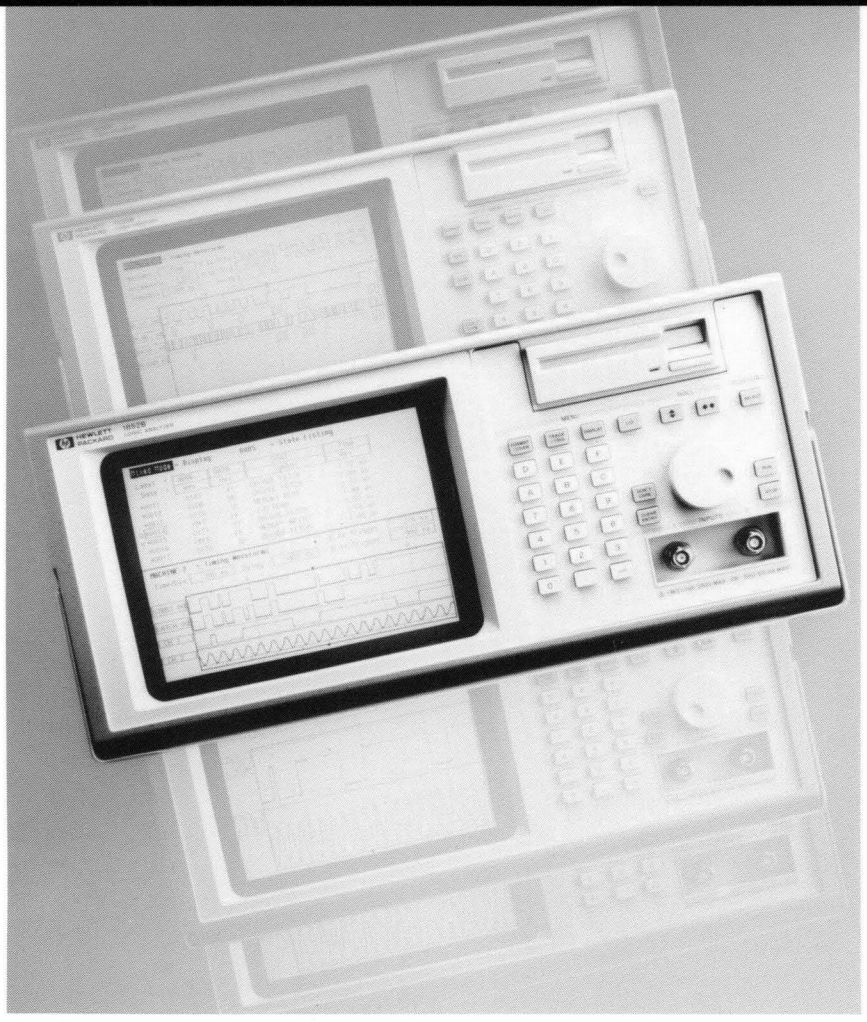
HP 1652B/HP 1653B
Logic Analyzers

Programming Reference



HEWLETT
PACKARD

Programming Reference

HP 1652B/HP 1653B Logic Analyzers



©Copyright Hewlett-Packard Company 1989

Manual Number 01652-90903

Printed in the U.S.A. December 1989

Product Warranty

This Hewlett-Packard product has a warranty against defects in material and workmanship for a period of three years from date of shipment. During warranty period, Hewlett-Packard Company will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard. However, warranty service for products installed by Hewlett-Packard and certain other products designated by Hewlett-Packard will be performed at the Buyer's facility at no charge within the Hewlett-Packard service travel area. Outside Hewlett-Packard service travel areas, warranty service will be performed at the Buyer's facility only upon Hewlett-Packard's prior agreement and the Buyer shall pay Hewlett-Packard's round trip travel expenses.

For products returned to Hewlett-Packard for warranty service, the Buyer shall prepay shipping charges to Hewlett-Packard and Hewlett-Packard shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Hewlett-Packard from another country.

Hewlett-Packard warrants that its software and firmware designated by Hewlett-Packard for use with an instrument will execute its programming instructions when properly installed on that instrument. Hewlett-Packard does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HEWLETT-PACKARD SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Exclusive Remedies THE REMEDIES PROVIDED HEREIN ARE THE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

Assistance Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Certification Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Safety This product has been designed and tested according to International Safety Requirements. To ensure safe operation and to keep the product safe, the information, cautions, and warnings in this manual must be heeded.

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

A software code may be printed before the date; this indicates the version level of the software product at the time of the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

Edition 1 December 1989 01652-90903

List of Effective Pages

The List of Effective Pages gives the data of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition will have the date the changes were made printed on the bottom of the page. If an update is incorporated when a new edition of the manual is printed, the change dates are removed from the bottom of the pages and the new edition date is listed in Printing History and on the title page.

Pages	Effective Date
All	December 1989

Contents

Chapter 1 Introduction to Programming an Instrument

Introduction 1-1
About This Manual 1-1
Programming Syntax 1-2
Talking to the Instrument 1-2
Instruction Syntax 1-2
Output Command 1-3
Device Address 1-3
Instructions 1-3
Instruction Header 1-3
White Space 1-4
Instruction Parameters 1-4
Header Types 1-4
Combining Commands from the Same Subsystem 1-5
Duplicate Keywords 1-5
Query Usage 1-6
Program Header Options 1-7
Parameter Syntax Rules 1-7
Instruction Terminator 1-9
Selecting Multiple Subsystems 1-9
Programming an Instrument 1-10
Initialization 1-10
Example Program 1-11
Program Overview 1-11
Receiving Information from the Instrument 1-11
Response Header Options 1-12
Response Data Formats 1-13
String Variables 1-14
Numeric Base 1-15
Numeric Variables 1-15
Definite-Length Block Response Data 1-16
Multiple Queries 1-17
Instrument Status 1-17

Chapter 2 Programming Over HP-IB

Introduction 2-1
Interface Capabilities 2-1
Command and Data Concepts 2-1
Addressing 2-1
Communicating Over the HP-IB Bus (HP 9000 Series 200/300 Controller) 2-2
Local, Remote, and Local Lockout 2-2
Bus Commands 2-3
  Device Clear 2-3
  Group Execute Trigger (GET) 2-3
  Interface Clear (IFC) 2-3

Chapter 3 Programming Over RS-232C

Introduction 3-1
Interface Operation 3-1
Cables 3-2
Minimum Three-Wire Interface with Software Protocol 3-2
Extended Interface with Hardware Handshake 3-3
Cable Example 3-4
Configuring the Instrument Interface 3-5
Interface Capabilities 3-5
  Protocol 3-5
  Data Bits 3-6
Communicating Over the RS-232C Bus (HP 9000 Series 200/300 Controller) 3-6
Lockout Command 3-7

Chapter 4 Programming and Documentation Conventions

Introduction 4-1
Truncation Rule 4-1
Infinity Representation 4-2
Sequential and Overlapped Commands 4-2
Response Generation 4-2
Syntax Diagrams 4-2
Notation Conventions and Definitions 4-3
The Command Tree 4-4

Command Types 4-4
Tree Traversal Rules 4-4
Examples 4-5
Command Set Organization 4-10
Subsystems 4-10
Program Examples 4-11

Chapter 5 Common Commands

Introduction 5-1
*CLS 5-3
*ESE 5-4
*ESR 5-6
*IDN 5-8
*OPC 5-9
*RST 5-10
*SRE 5-11
*STB 5-13
*WAI 5-15

Chapter 6 System Commands

Introduction 6-1
ARMBnc 6-4
DATA 6-5
Logic Analyzer Block Data 6-8
Section Header Description 6-8
Section Data 6-8
Data Preamble Description 6-8
Acquisition Data Description 6-11
Oscilloscope Block Data 6-18
Oscilloscope Data Section 6-18
Section Header Description 6-18
Section Data 6-18
Oscilloscope Display Data Section 6-19
DSP 6-20
ERRor 6-21
HEADer 6-22
KEY 6-23
LER 6-25
LOCKout 6-26

LONGform 6-27
MENU 6-28
MESE 6-29
MESR 6-31
PPOWer 6-33
PRINt 6-34
RMODe 6-35
SETup 6-36
STARt 6-38
STOP 6-39

Chapter 7 **MMEMory Subsystem**

Introduction 7-1
AUToload 7-4
CATalog 7-5
COPY 7-6
DOWNload 7-7
INITialize 7-8
LOAD 7-9
LOAD 7-10
PACK 7-11
PURGe 7-12
REName 7-13
STORe 7-14
UPLoad 7-15

Chapter 8 **DLISt Subsystem**

Introduction 8-1
DLISt 8-2
COLumn 8-3
LINE 8-5

Chapter 9 WLISt Subsystem

Introduction 9-1
WLISt 9-2
OSTate 9-3
XSTate 9-4
OTIMe 9-5
XTIMe 9-6

Chapter 10 MACHine Subsystem

Introduction 10-1
MACHine 10-3
ARM 10-4
ASSign 10-5
AUToscale 10-6
NAME 10-7
TYPE 10-8

Chapter 11 SFORmat Subsystem

Introduction 11-1
SFORmat 11-3
CLOCk 11-4
CPERiod 11-5
LABel 11-6
MASTer 11-8
REMove 11-9
SLAVe 11-10
THReshold 11-11

Chapter 12 STRace Subsystem

Introduction 12-1
STRace 12-4
BRANch 12-5
FIND 12-8
PREStore 12-10
RANGe 12-12

RESTart 12-14
SEQuence 12-16
STORe 12-17
TAG 12-19
TERM 12-21

Chapter 13 **SLISt Subsystem**

Introduction 13-1
SLISt 13-5
COLumn 13-6
DATA 13-8
LINE 13-9
MMODe 13-10
OPATtern 13-11
OSEarch 13-13
OSTate 13-14
OTAG 13-15
RUNTil 13-16
TAVerage 13-18
TMAXimum 13-19
TMINimum 13-20
VRUNs 13-21
XOTag 13-22
XPATtern 13-23
XSEarch 13-25
XSTate 13-26
XTAG 13-27

Chapter 14 **SWAVeform Subsystem**

Introduction 14-1
SWAVeform 14-3
ACCumulate 14-4
DELay 14-5
INSert 14-6
RANGe 14-7
REMove 14-8

Chapter 15 SCHart Subsystem

Introduction 15-1
SCHart 15-3
ACCumulate 15-4
HAXis 15-5
VAXis 15-7

Chapter 16 COMPare Subsystem

Introduction 16-1
COMPare 16-3
CMASk 16-4
COPY 16-5
DATA 16-6
FIND 16-8
RANGe 16-9
RUNTil 16-10

Chapter 17 TFORmat Subsystem

Introduction 17-1
TFORmat 17-2
LABel 17-3
REMove 17-5
THReshold 17-6

Chapter 18 TTRace Subsystem

Introduction 18-1
TTRace 18-3
AMODe 18-4
DURation 18-5
EDGE 18-6
GLITch 18-8
PATTern 18-9

Chapter 19 TWAVeform Subsystem

Introduction 19-1
TWAVeform 19-5
ACCumulate 19-6
DELay 19-7
INSert 19-8
MMODe 19-9
OCONdition 19-10
OPATtern 19-11
OSEarch 19-13
OTIMe 19-14
RANGe 19-15
REMove 19-16
RUNTil 19-17
SPERiod 19-19
TAVerage 19-20
TMAXimum 19-21
TMINimum 19-22
VRUNs 19-23
XCONdition 19-24
XOTime 19-25
XPATtern 19-26
XSEarch 19-28
XTIMe 19-29

Chapter 20 SYMBol Subsystem

Introduction 20-1
SYMBol 20-3
BASE 20-4
PATTern 20-5
RANGe 20-6
REMove 20-7
WIDTh 20-8

Chapter 21 **SCOPe Subsystem**

Introduction 21-1
SCOPe 21-3
ARM 21-4
AUToscale 21-5
SMODe 21-6

Chapter 22 **CHANnel Subsystem**

Introduction 22-1
CHANnel 22-3
COUPling 22-4
OFFSet 22-5
PROBe 22-6
RANGe 22-7

Chapter 23 **TRIGger Subsystem**

Introduction 23-1
The Edge
Trigger Mode 23-1
The Immediate Trigger Mode 23-1
TRIGger 23-3
LEVEL 23-4
MODE 23-5
SLOPe 23-6
SOURce 23-7

Chapter 24 **ACQuire Subsystem**

Introduction 24-1
Acquisition Type Normal 24-2
Acquisition Type Average 24-2
ACQuire 24-3
COUNt 24-4
TYPE 24-5

Chapter 25 TIMebase Subsystem

Introduction 25-1
TIMebase 25-2
DELAY 25-3
MODE 25-4
RANGe 25-6

Chapter 26 WAVeform Subsystem

Introduction 26-1
Waveform
Record 26-3
Data Acquisition Types 26-3
Normal Mode 26-3
Average Mode 26-3
Format for Data Transfer 26-4
BYTE Format 26-4
WORD Format 26-5
ASCII Format 26-5
Data Conversion 26-6
Conversion from Data Value to Voltage 26-6
Conversion from Data Value to Time 26-6
Conversion from Data Value to Trigger Point 26-6
WAVeform 26-7
COUNt 26-8
DATA 26-9
FORMat 26-10
POINts 26-11
PREAmble 26-12
RECord 26-13
SOURce 26-14
TYPE 26-15
VALid 26-16
XINCrement 26-17
XORigin 26-18
XREFerence 26-19
YINCrement 26-20
YORigin 26-21
YREFerence 26-22

Chapter 27 MEASure Subsystem

Introduction 27-1
- Frequency 27-2
- Period 27-2
- Peak-to-Peak 27-2
- Positive Pulse Width 27-2
- Negative Pulse Width 27-2
- Risetime 27-2
- Falltime 27-2
- Preshoot and Overshoot 27-2
- Preshoot 27-2
- Overshoot 27-2
  - MEASure 27-4
  - ALL 27-5
  - FALLTime 27-6
  - FREQuency 27-7
  - NWIDth 27-8
  - OVERShoot 27-9
  - PERiod 27-10
  - PRESHoot 27-11
  - PWIDth 27-12
  - RISETIme 27-13
  - SOURce 27-14
  - VAMPlitude 27-15
  - VBASe 27-16
  - VMAX 27-17
  - VMIN 27-18
  - VPP 27-19
  - VTOP 27-20

Syntax Overview A-5
Device Listening Syntax A-8
Device Talking Syntax A-21
Common Commands A-27

Appendix B **Status Reporting**

Introduction B-1
Event Status Register B-3
Service Request Enable Register B-3
Bit Definitions B-3
Key Features B-4
Serial Poll B-6
Using Serial Poll (HP-IB) B-6

Appendix C **Error Messages**

Device Dependent Errors C-1
Command Errors C-2
Execution Errors C-3
Internal Errors C-4
Query Errors C-5

Index

Introduction to Programming an Instrument

1

Introduction

This chapter introduces you to the basics of remote programming. The programming instructions explained in this book conform to the IEEE 488.2 Standard Digital Interface for Programmable Instrumentation. These programming instructions provide a means of remotely controlling the HP 1652B/53B. There are three general categories of use. You can:

- Set up the instrument and start measurements
- Retrieve setup information and measurement results
- Send measurement data to the instrument

The instructions listed in this manual give you access to the measurements and front panel features of the HP 1652B/53B. The complexity of your programs and the tasks they accomplish are limited only by your imagination. This programming reference is designed to provide a concise description of each instruction.

About This Manual

This manual is organized in 27 chapters. Chapter 1 is divided into two sections. The first section (pages 2 through 9) concentrates on program syntax, and the second section (pages 10 through 17) discusses programming an instrument. Read either chapter 2, "Programming Over HP-IB," or chapter 3, "Programming Over RS-232C" for information concerning the physical connection between the HP 1652B/53B and your controller. Chapter 4, "Programming and Documentation Conventions," gives an overview of all instructions and also explains the notation conventions used in our syntax definitions and examples. The remaining chapters 5 through 27 are used to explain each group of instructions.

Programming Syntax

Talking to the Instrument

In general, computers acting as controllers communicate with the instrument by sending and receiving messages over a remote interface, such as HP-IB or RS-232C. Instructions for programming the HP 1652B/53B will normally appear as ASCII character strings embedded inside the output statements of a "host" language available on your controller. The host language's input statements are used to read in responses from the HP 1652B/53B.

For example, HP 9000 Series 200/300 BASIC uses the OUTPUT statement for sending commands and queries to the HP 1652B/53B. After a query is sent, the response is usually read in using the ENTER statement. All programming examples in this manual are presented in BASIC. The following BASIC statement sends a command which causes the HP 1652B/53B's machine 1 to be a state analyzer:

```
OUTPUT XXX;":MACHINE1:TYPE STATE"  <terminator>
```

Each part of the above statement is explained in the following pages.

Instruction Syntax

To program the instrument remotely, you must have an understanding of the command format and structure expected by the instrument. The IEEE 488.2 syntax rules govern how individual elements such as headers, separators, parameters and terminators may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses will be formatted. Figure 1-1 shows the main syntactical parts of a typical program statement.

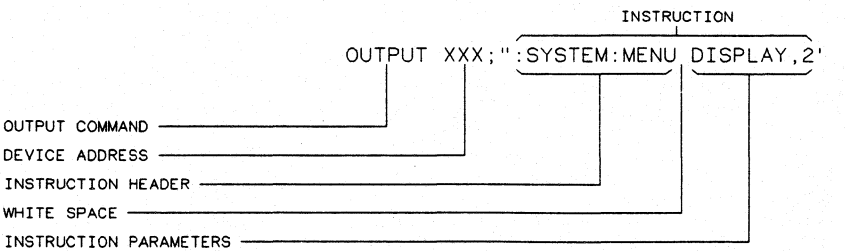


Figure 1-1. Program Message Syntax

Output Command

The output command is entirely dependant on the language you choose to use. Throughout this manual HP 9000 Series 200/300 BASIC 4.0 is used in the programming examples. People using another language will need to find the equivalents of BASIC commands like OUTPUT, ENTER and CLEAR in order to convert the examples. The instructions for the HP 1652B/53B are always shown between the double-quotes.

Device Address

The location where the device address must be specified is also dependent on the host language which you are using. In some languages, this could be specified outside the output command. In BASIC, this is always specified after the keyword OUTPUT. The examples in this manual use a generic address of XXX. When writing programs, the number you use will depend on the cable you use in addition to the actual address. If you are using an HP-IB, see chapter 2. RS-232C users should refer to chapter 3, "Programming Over RS-232C."

Instructions

Instructions (both commands and queries) normally appear as a string embedded in a statement of your host language, such as BASIC, Pascal or C. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>. There are only five instructions which use block data.

Instructions are composed of two main parts: The header, which specifies the command or query to be sent; and the parameters, which provide additional data needed to clarify the meaning of the instruction.

Instruction Header

The instruction header is one or more keywords separated by colons (:). The command tree in figure 4-1 illustrates how all the keywords can be joined together to form a complete header (see chapter 4, "Programming and Documentation Conventions").

The example in figure 1-1 shows a command. Queries are indicated by adding a question mark (?) to the end of the header. Many instructions can be used as either commands or queries, depending on whether or not you have included the question mark. The command and query forms of an instruction usually have different parameters. Many queries do not use any parameters.

When you look up a query in this programming reference, you'll find a paragraph labeled "Returned Format" under the one labeled "Query Syntax." The syntax definition by "Returned format" will always show the instruction header in square brackets, like [:SYSTem:MENU]. What this

really means is that the text between the brackets is optional, but it's also a quick way to see what the header looks like.

White Space

White space is used to separate the instruction header from the instruction parameters. If the instruction does not use any parameters, you do not need to include any white space. White space is defined as one or more spaces. ASCII defines a space to be character 32 (in decimal). Tabs can be used only if your controller first converts them to space characters before sending the string to the instrument.

Instruction Parameters

Instruction parameters are used to clarify the meaning of the command or query. They provide necessary data, such as whether a function should be on or off, which waveform is to be displayed, or which pattern is to be looked for. Each instruction's syntax definition shows the parameters, as well as the values they accept. This chapter's "Parameter Syntax Rules" section has all of the general rules about acceptable values.

When there is more than one parameter they are separated by commas (,). You are allowed to add spaces around the commas.

Header Types

There are three types of headers: Simple Command; Compound Command; and Common Command.

Simple Command Header. Simple command headers contain a single keyword. START and STOP are examples of simple command headers typically used in this instrument. The syntax is:

<function> <terminator>

When parameters (indicated by <data>) must be included with the simple command header (for example, :RMODE SINGLE) the syntax is:

<function> <white space> <data> <terminator>

Compound Command Header. Compound command headers are a combination of two or more program keywords. The first keyword selects the subsystem, and the last keyword selects the function within that subsystem. Sometimes you may need to list more than one subsystem before being allowed to specify the function. The keywords within the compound header are separated by colons. For example:

To execute a single function within a subsystem, use the following:

:<subsystem>:<function> <white space> <data> <terminator>

(For example :SYSTEM:LONGFORM ON)

To traverse down a level of a subsystem to execute a subsystem within that subsystem:

:<subsystem>:<subsystem>:<function> <white space> <data> <terminator>

(For example :MMEMORY:LOAD:CONFIG "FILE__")

Common Command Header. Common command headers control IEEE 488.2 functions within the instrument (such as clear status, etc.). Their syntax is:

*<command header> <terminator>

No space or separator is allowed between the asterisk and the command header. *CLS is an example of a common command header.

Combining Commands from the Same Subsystem

To execute more than one function within the same subsystem a semi-colon (;) is used to separate the functions:

:<subsystem>:<function> <white space> <data>;
<function> <white space> <data> <terminator>

(For example :SYSTEM:LONGFORM ON;HEADER ON)

Duplicate Keywords

Identical function keywords can be used for more than one subsystem. For example, the function keyword MMODE may be used to specify the marker mode in the subsystem for state listing or the timing waveforms:

:SLIST:MMODE PATTERN - sets the marker mode to pattern in the state listing.

:TWAVEFORM:MMODE TIME - sets the marker mode to time in the timing waveforms.

SLIST and TWAVEFORM are subsystem selectors and determine which marker mode is being modified.

Query Usage Command headers immediately followed by a question mark (?) are queries. After receiving a query, the instrument interrogates the requested function and places the response in its output queue. The output message remains in the queue until it is read or another command is issued. When read, the message is transmitted across the bus to the designated listener (typically a controller). For example, the logic analyzer query :MACHINE1:TWAVEFORM:RANGE? places the current seconds per division full scale range for machine 1 in the output queue. In BASIC, the input statement

```
ENTER XXX; Range
```

passes the value across the bus to the controller and places it in the variable Range.

Query commands are used to find out how the instrument is currently configured. They are also used to get results of measurements made by the instrument. For example, the command

:MACHINE1:TWAVEFORM:XOTIME?

instructs the instrument to place the X to O time in the output queue.

Note 

The output queue must be read before the next program message is sent. For example, when you send the query :TWAVEFORM:XOTIME? you must follow that with an input statement. In BASIC, this is usually done with an ENTER statement.

Sending another command before reading the result of the query will cause the output buffer to be cleared and the current response to be lost. This will also generate a "QUERY UNTERMINATED" error in the error queue.

Program Header Options

Program headers can be sent using any combination of uppercase or lowercase ASCII characters. Instrument responses, however, are always returned in uppercase.

Both program command and query headers may be sent in either longform (complete spelling), shortform (abbreviated spelling), or any combination of longform and shortform. Either of the following examples turns on the headers and longform.

```
OUTPUT XXX;":SYSTEM:HEADER ON;LONGFORM ON"    - longform
OUTPUT XXX;":SYST:HEAD ON;LONG ON"          - shortform
```

Programs written in longform are easily read and are almost self-documenting. The shortform syntax conserves the amount of controller memory needed for program storage and reduces the amount of I/O activity.

Note 

The rules for shortform syntax are shown in chapter 4 "Programming and Documentation Conventions."

Parameter Syntax Rules

There are three main types of data which are used in parameters. They are numeric, string, and keyword. A fourth type, block data, is used only for five instructions: the DATA and SETup instructions in the SYSTem subsystem (see chapter 6); the CATalog, UPLoad, and DOWNload instructions in the MMEMory subsystem (see chapter 7). These syntax rules also show how data may be formatted when sent back from the HP 1652B/53B as a response.

The parameter list always follows the instruction header and is separated from it by white space. When more than one parameter is used, they are separated by commas. You are allowed to include one or more spaces around the commas, but it is not mandatory.

Numeric data. For numeric data, you have the option of using exponential notation or using suffixes to indicate which unit is being used. Tables A-1 and A-2 in appendix A list all available suffixes. Do not combine an exponent with a unit. The following numbers are all equal: 28 = 0.28E2 = 280e-1 = 28000m = 0.028K.

The base of a number is shown with a prefix. The available bases are binary (#B), octal (#Q), hexadecimal (#H) and decimal (default). For example, #B11100 = #Q34 = #H1C = 28. You may not specify a base in conjunction with either exponents or unit suffixes. Additionally, negative numbers must be expressed in decimal.

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part would be ignored, truncating the number. Numeric parameters which accept fractional values are called real numbers.

All numbers are expected to be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character "9" (which is 57, or 0011 1001 in binary). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48 and 50). This is taken care of automatically when you include the entire instruction in a string.

String data. String data may be delimited with either single (') or double (") quotes. String parameters representing labels are case-sensitive. For instance, the labels "Bus A" and "bus a" are unique and should not be used indiscriminately. Also pay attention to the presence of spaces, since they act as legal characters just like any other. So the labels "In" and " In" are also two separate labels.

Keyword data. In many cases a parameter must be a keyword. The available keywords are always included with the instruction's syntax definition. When sending commands, either the longform or shortform (if one exists) may be used. Upper-case and lower-case letters may be mixed freely. When receiving responses, upper-case letters will be used exclusively. The use of longform or shortform in a response depends on the setting you last specified via the SYSTem:LONGform command (see chapter 6).

Instruction Terminator

An instruction is executed after the instruction terminator is received. The terminator is the NL (New Line) character. The NL character is an ASCII linefeed character (decimal 10).

Note

The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

<instruction header> <data>;:<instruction header> <data> <terminator>

```
:MACHINE1:ASSIGN2;:SYSTEM:HEADERS ON
```

Note

Multiple commands may be any combination of simple, compound and common commands.

Programming an Instrument

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. BASIC provides a CLEAR command which clears the interface buffer. If you're using HP-IB, CLEAR will also reset the HP 1652B/53B's parser. The parser is the program which reads in the instructions which you send it.

After clearing the interface, load a predefined configuration file from the disk to preset the instrument to a known state. For example:

```
OUTPUT XXX;":MMEMORY:LOAD:CONFIG 'DEFAULT__'"
```

This BASIC statement would load the configuration file "DEFAULT__" (if it exists) into the HP 1652B/53B. Refer to the chapter "MMEMory Subsystem" for more information on the LOAD command.

Note 

Refer to your controller manual and programming language reference manual for information on initializing the interface.

Example Program

This program demonstrates the basic command structure used to program the HP 1652B/53B.

```
10 CLEAR XXX                              !Initialize instrument interface
20 OUTPUT XXX;":SYSTEM:HEADER ON"                    !Turn headers on
30 OUTPUT XXX;":SYSTEM:LONGFORM ON"                 !Turn longform on
40 OUTPUT XXX;":MMEM:LOAD:CONFIG 'TEST_E'"     !Load configuration file
50 OUTPUT XXX;":MENU FORMAT,1"         !Select Format menu for machine 1
60 OUTPUT XXX;":RMODE SINGLE"                        !Select run mode
70 OUTPUT XXX;":START"                             !Run the measurement
```

Program Overview

Line 10 initializes the instrument interface to a known state
Lines 20 and **30** turn the headers and longform on.
Line 40 loads the configuration file "TEST_E" from the disc drive.
Line 50 displays the Format menu for machine 1.
Lines 60 and **70** tell the analyzer to run the measurement configured by the file "TEST_E" one time.

Receiving Information from the Instrument

After receiving a query (command header followed by a question mark), the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the message is transmitted across the bus to the designated listener (typically a controller). The input statement for receiving a response message from an instrument's output queue typically has two parameters;the device address and a format specification for handling the response message. For example, to read the result of the query command :SYSTEM:LONGFORM? you could execute the BASIC statement:

```
ENTER XXX; Setting
```

where XXX represents the address of your device. This would enter the current setting for the longform command in the numeric variable *Setting*.

Note

All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :MACHINE1:ASSIGN?, you must follow that query with an input statement. In BASIC, this is usually done with an ENTER statement.

The format specification for handling the response messages is dependent on both the controller and the programming language.

Response Header Options

The format of the returned ASCII string depends on the current settings of the SYSTEM HEADER and LONGFORM commands. The general format is:

<instruction header> <space> <data> <terminator>

The header identifies the data that follows (the parameters) and is controlled by issuing a :SYSTEM:HEADER ON/OFF command. If the state of the header command is OFF, only the data is returned by the query.

The format of the header is controlled by the :SYSTEM:LONGFORM ON/OFF command. If longform is OFF , the header will be in its shortform and the header will vary in length depending on the particular query. The separator between the header and the data always consists of one space.

The following examples show some possible responses for a :MACHINE1:SFORMAT:THRESHOLD2? query:

- with HEADER OFF:
  <data> <terminator>
- with HEADER ON and LONGFORM OFF:
  :MACH1:SFOR:THR2 <space> <data> <terminator>
- with HEADER ON and LONGFORM ON:
  :MACHINE1:SFORMAT:THRESHOLD2 <space> <data> <terminator>

Note

A command or query may be sent in either longform or shortform, or in any combination of longform and shortform. The HEADER and LONGFORM commands only control the format of the returned data and have no effect on the way commands are sent.

Refer to the chapter "System Commands" for information on turning the HEADER and LONGFORM commands on and off.

Response Data Formats

Both numbers and strings are returned as a series of ASCII characters, as described in the following sections. Keywords in the data are returned in the same format as the header, as specified by the LONGform command. Like the headers, the keywords will always be in upper-case.

The following are possible responses to the "MACHINE1: TFORMAT: LAB? 'ADDR' " query.

MACHINE1:TFORMAT:LABEL "ADDR ",19,POSITIVE<terminator> (Header on; Longform on)

MACH1:TFOR:LAB "ADDR ",19,POS<terminator> (Header on; Longform off)

"ADDR ",19,POSITIVE<terminator> (Header off; Longform on)

"ADDR ",19,POS<terminator> (Header off; Longform off)

Note

Refer to the individual commands in this manual for information on the format (alpha or numeric) of the data returned from each query.

String Variables Since there are so many ways to code numbers, the HP 1652B/53B handles almost all data as ASCII strings. Depending on your host

The following example shows logic analyzer data being returned to a string variable with headers off:

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 DIM Rang$[30]
30 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
40 ENTER XXX;Rang$
50 PRINT Rang$
60 END
```

After running this program, the controller displays:

+1.00000E-05

Numeric Base

Most numeric data will be returned in the same base as shown on screen. When the prefix #B precedes the returned data, the value is in the binary base. Likewise, #Q is the octal base and #H is the hexadecimal base. If no prefix precedes the returned numeric data, then the value is in the decimal base.

Numeric Variables

If your host language can convert from ASCII to a numeric format, then you can use numeric variables. Turning off the response headers will help you avoid accidently trying to convert the header into a number.

The following example shows logic analyzer data being returned to a numeric variable.

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
30 ENTER XXX;Rang
40 PRINT Rang
50 END
```

This time the format of the number (such as whether or not exponential notation is used) is dependant upon your host language. In BASIC, the output would look like:

1.E-5

Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 80 bytes of data, the syntax would be:

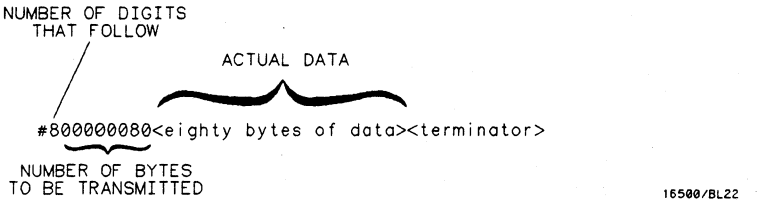


Figure 1-2. Definite-length Block Response Data

The "8" states the number of digits that follow, and "00000080" states the number of bytes to be transmitted.

Note Indefinite-length block data is not supported on the HP1652B/53B.

Multiple Queries

You can send multiple queries to the instrument within a single program message, but you must also read them back within a single program message. This can be accomplished by either reading them back into a string variable or into multiple numeric variables. For example, you could read the result of the query :SYSTEM:HEADER?;LONGFORM? into the string variable Results$ with the command:

```
ENTER XXX; Results$
```

When you read the result of multiple queries into string variables, each response is separated by a semicolon. For example, the response of the query :SYSTEM:HEADER?:LONGFORM? with HEADER and LONGFORM on would be:

```
:SYSTEM:HEADER 1;:SYSTEM:LONGFORM 1
```

If you do not need to see the headers when the numeric values are returned, then you could use following program message to read the query :SYSTEM:HEADERS?;LONGFORM? into multiple numeric variables:

```
ENTER XXX; Result1, Result2
```

Note 

When you are receiving numeric data into numeric variables, the headers should be turned off. Otherwise the headers may cause misinterpretation of returned data.

Instrument Status

Status registers track the current status of the instrument. By checking the instrument status, you can find out whether an operation has been completed, whether the instrument is receiving triggers, and more. Appendix B, "Status Reporting," explains how to check the status of the instrument.

Programming Over HP-IB 2

Introduction

This section describes the interface functions and some general concepts of the HP-IB. In general, these functions are defined by IEEE 488.1 (HP-IB bus standard). They deal with general bus management issues, as well as messages which can be sent over the bus as bus commands.

Interface Capabilities

The interface capabilities of the HP 1652B/53B, as defined by IEEE 488.1 are SH1, AH1, T5, TE0, L3, LE0, SR1, RL1, PP0, DC1, DT1, C0, and E2.

Command and Data Concepts

The HP-IB has two modes of operation: command mode and data mode. The bus is in command mode when the ATN line is true. The command mode is used to send talk and listen addresses and various bus commands, such as a group execute trigger (GET). The bus is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. These device-dependent messages include all of the instrument commands and responses found in chapters 5 through 27 of this manual.

Addressing

By using the front-panel I/O and SELECT keys, the HP-IB interface can be placed in either talk only mode "Printer connected to HP-IB" or addressed talk/listen mode "Controller connected to HP-IB" (see "I/O Port Configuration" in Chapter 5 of the *HP 1652B/HP 1653B Front-Panel Reference* manual. Talk only mode must be used when you want the instrument to talk directly to a printer without the aid of a controller. Addressed talk/listen mode is used when the instrument will operate in conjunction with a controller. When the instrument is in the addressed talk/listen mode, the following is true:

- Each device on the HP-IB resides at a particular address ranging from 0 to 30.
- The active controller specifies which devices will talk, and which will listen.
- An instrument, therefore, may be talk addressed, listen addressed, or unaddressed by the controller.

If the controller addresses the instrument to talk, it will remain configured to talk until it receives an interface clear message (IFC), another instrument's talk address (OTA), its own listen address (MLA), or a universal untalk (UNT) command.

If the controller addresses the instrument to listen, it will remain configured to listen until it receives an interface clear message (IFC) its own talk address (MTA), or a universal unlisten (UNL) command.

Communicating Over the HP-IB Bus (HP 9000 Series 200/300 Controller)

Since HP-IB can address multiple devices through the same interface card, the device address passed with the program message must include not only the correct instrument address, but also the correct interface code.

Interface Select Code (Selects Interface). Each interface card has its own interface select code. This code is used by the controller to direct commands and communications to the proper interface. The default is always "7" for HP-IB controllers.

Instrument Address (Selects Instrument). Each instrument on the HP-IB port must have a unique instrument address between decimal 0 and 30. The device address passed with the program message must include not only the correct instrument address, but also the correct interface select code.

DEVICE ADDRESS = (Interface Select Code) X 100 + (Instrument Address)

For example, if the instrument address for the HP 1652B/53B is 4 and the interface select code is 7, when the program message is passed, the routine performs its function on the instrument at device address 704.

Local, Remote, and Local Lockout

The local, remote, and remote with local lockout modes may be used for various degrees of front-panel control while a program is running. The instrument will accept and execute bus commands while in local mode, and the front panel will also be entirely active. If the HP 1652B/53B is in remote mode, the instrument will go from remote to local with any front panel activity. In remote with local lockout mode, all controls (except the power switch) are entirely locked out. Local control can only be restored by the controller.

Note Cycling the power will also restore local control, but this will also reset certain HP-IB states.

The instrument is placed in remote mode by setting the REN (Remote Enable) bus control line true, and then addressing the instrument to listen. The instrument can be placed in local lockout mode by sending the local lockout (LLO) command (see SYSTem:LOCKout in chapter 6). The instrument can be returned to local mode by either setting the REN line false, or sending the instrument the go to local (GTL) command.

Bus Commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions which are taken when these commands are received by an instrument.

Device Clear

The device clear (DCL) or selected device clear (SDC) commands clear the input and output buffers, reset the parser, clear any pending commands, and clear the Request-OPC flag.

Group Execute Trigger (GET)

The group execute trigger command will cause the same action as the START command for Group Run: the instrument will acquire data for the active waveform and listing display(s).

Interface Clear (IFC)

This command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system controller.

Programming Over RS-232C 3

Introduction

This section describes the interface functions and some general concepts of the RS-232C. The RS-232C interface on this instrument is Hewlett-Packard's implementation of EIA Recommended Standard RS-232C, "Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange." With this interface, data is sent one bit at a time and characters are not synchronized with preceding or subsequent data characters. Each character is sent as a complete entity without relationship to other events.

Interface Operation

The HP 1652B/53B can be programmed with a controller over RS-232C using either a minimum three-wire or extended hardwire interface. The operation and exact connections for these interfaces are described in more detail in the following sections. When you are programming an HP 1652B/53B over RS-232C with a controller, you are normally operating directly between two DTE (Data Terminal Equipment) devices as compared to operating between a DTE device and a DCE (Data Communications Equipment) device.

When operating directly between two DTE devices, certain considerations must be taken into account. For three-wire operation, XON/XOFF must be used to handle protocol between the devices. For extended hardwire operation, protocol may be handled either with XON/XOFF or by manipulating the CTS and RTS lines of the RS-232C link. For both three-wire and extended hardwire operation, the DCD and DSR inputs to the HP 1652B/53B must remain high for proper operation.

With extended hardwire operation, a high on the CTS input allows the HP 1652B/53B to send data and a low on this line disables the HP 1652B/53B data transmission. Likewise, a high on the RTS line allows the controller to send data and a low on this line signals a request for the controller to disable data transmission. Since three-wire operation has no control over the CTS input, internal pull-up resistors in the HP 1652B/53B assure that this line remains high for proper three-wire operation.

Cables

Selecting a cable for the RS-232C interface is dependent on your specific application. The following paragraphs describe which lines of the HP 1652B/53B are used to control the operation of the RS-232C relative to the HP 1652B/53B. To locate the proper cable for your application, refer to the reference manual for your controller. This manual should address the exact method your controller uses to operate over the RS-232C bus.

Minimum Three-Wire Interface with Software Protocol

With a three-wire interface, the software (as compared to interface hardware) controls the data flow between the HP 1652B/53B and the controller. This provides a much simpler connection between devices since you can ignore hardware handshake requirements. The HP 1652B/53B uses the following connections on its RS-232C interface for three-wire communication:

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from HP 1652B/53B)
- Pin 3 RD (Receive Data into HP 1652B/53B)

The TD (Transmit Data) line from the HP 1652B/53B must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the HP 1652B/53B must connect to the TD line on the controller. Internal pull-up resistors in the HP 1652B/53B assure the DCD, DSR, and CTS lines remain high when you are using a three-wire interface.

Note 

The three-wire interface provides no hardware means to control data flow between the controller and the HP 1652B/53B. XON/OFF protocol is the only means to control this data flow.

Extended Interface with Hardware Handshake

With the extended interface , both the software and the hardware can control the data flow between the HP 1652B/53B and the controller. This allows you to have more control of data flow between devices. The HP 1652B/53B uses the following connections on its RS-232C interface for extended interface communication:

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from HP 1652B/53B)
- Pin 3 RD (Receive Data into HP 1652B/53B)

The additional lines you use depends on your controller's implementation of the extended hardwire interface.

- Pin 4 RTS (Request To Send) is an output from the HP 1652B/53B which can be used to control incoming data flow.
- Pin 5 CTS (Clear To Send) is an input to the HP 1652B/53B which controls data flow from the HP 1652B/53B.
- Pin 6 DSR (Data Set Ready) is an input to the HP 1652B/53B which controls data flow from the HP 1652B/53B within two bytes.
- Pin 8 DCD (Data Carrier Detect) is an input to the HP 1652B/53B which controls data flow from the HP 1652B/53B within two bytes.
- Pin 20 DTR (Data Terminal Ready) is an output from the HP 1652B/53B which is enabled as long as the HP 1652B/53B is turned on.

The TD (Transmit Data) line from the HP 1652B/53B must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the HP 1652B/53B must connect to the TD line on the controller.

The RTS (Request To Send), is an output from the HP 1652B/53B which can be used to control incoming data flow. A true on the RTS line allows the controller to send data and a false on this line signals a request for the controller to disable data transmission.

The CTS (Clear To Send), DSR (Data Set Ready), and DCD (Data Carrier Detect) lines are inputs to the HP 1652B/53B which control data flow from the HP 1652B/53B (Pin 2). Internal pull-up resistors in the HP 1652B/53B assure the DCD and DSR lines remain high when they are not connected. If DCD or DSR are connected to the controller, the controller must keep these lines and the CTS line high to enable the HP 1652B/53B to send data to the controller. A low on any one of these lines will disable the HP 1652B/53B data transmission. Dropping the CTS line low during data transmission will stop HP 1652B/53B data transmission immediately. Dropping either the DSR or DCD line low during data transmission will stop HP 1652B/53B data transmission, but as many as two additional bytes may be transmitted from the HP 1652B/53B.

Cable Example

Figure 3-1 is an example of how to connect the HP 1652B/53B to the HP 98628A Interface card of an HP 9000 series 200/300 controller. For more information on cabling, refer to the reference manual for your specific controller.

Note

Since this example does not have the correct connections for hardware handshake, XON/XOFF protocol must be used when connecting the HP 1652B/53B as shown in figure 3-1

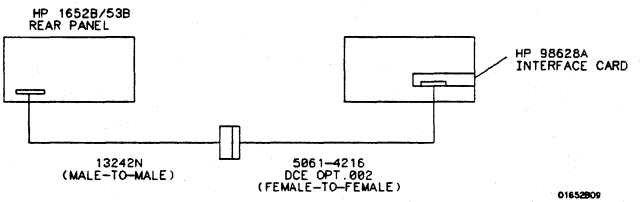


Figure 3-1. Cable Example

Configuring the Instrument Interface

The front-panel I/O menu key allows you access to the RS-232C Configuration menu where the RS-232C interface is configured.

If you are not familiar with how to configure the RS-232C interface, refer to the *HP 1652B/53B Front-panel Reference* manual.

Interface Capabilities

The baud rate, stop bits, parity, protocol, and data bits must be configured exactly the same for both the controller and the HP 1652B/53B to properly communicate over the RS-232C bus. The HP 1652B/53B RS-232C interface capabilities are listed below:

- Baud Rate: 110, 300, 600, 1200, 2400, 4800, 9600, or 19.2 k
- Stop Bits: 1, 1.5, or 2
- Parity: None, Odd, or Even
- Protocol: None or XON/XOFF
- Data Bits: 8

Protocol

NONE. With a three-wire interface, selecting NONE for the protocol does not allow the sending or receiving device to control data flow. No control over the data flow increases the possibility of missing data or transferring incomplete data.

With an extended hardwire interface, selecting NONE allows a hardware handshake to occur. With hardware handshake, hardware signals control data flow.

XON/XOFF. XON/XOFF stands for Transmit On/Transmit Off. With this mode the receiver (controller or HP 1652B/53B) controls data flow and can request that the sender (HP 1652B/53B or controller) stop data flow. By sending XOFF (ASCII 19) over its transmit data line, the receiver requests that the sender disables data transmission. A

Data Bits Data bits are the number of bits sent and received per character that represent the binary code of that character. Characters consist of either 7 or 8 bits, depending on the application. The HP 1652B/53B supports 8 bit only.

8 Bit Mode. Information is usually stored in bytes (8 bits at a time). With 8-bit mode, you can send and receive data just as it is stored, without the need to convert the data.

Note 

The controller and the HP 1652B/53B must be in the same bit mode to properly communicate over the RS-232C. This means that both the controller and the HP 1652B/53B must have the capability to send and receive 8 bit data.

For more information on the RS-232C interface, refer to the *HP 1652B/HP 1653B Front-Panel Reference* Manual. For information on RS-232C voltage levels and connector pinouts, refer to the *HP 1652B/53B Service Manual.*

Communicating Over the RS-232C Bus (HP 9000 Series 200/300 Controller)

Each RS-232C interface card has its own interface select code. This code is used by the controller to direct commands and communications to the proper interface by specifying the correct interface code for the device address.

Generally, the interface select code can be any decimal value between 0 and 31, except for those interface codes which are reserved by the controller for internal peripherals and other internal interfaces. This value can be selected through switches on the interface card. For more information, refer to the reference manual for your interface card or controller.

For example, if your RS-232C interface select code is 9, the device address required to communicate over the RS-232C bus is 9.

Lockout Command

To lockout the front panel controls use the SYSTem command LOCKout. When this function is on, all controls (except the power switch) are entirely locked out. Local control can only be restored by sending the command :LOCKout OFF. For more information on this command see the chapter "System Commands" in this manual.

Note 

Cycling the power will also restore local control, but this will also reset certain RS-232C states.

Programming and Documentation Conventions

4

Introduction

This section covers the programming conventions used in programming the instrument, as well as the documentations conventions used in this manual. This chapter also contains a detailed description of the command tree and command tree traversal.

Truncation Rule

The truncation rule for the keywords used in headers and parameters is:

If the longform has four or fewer characters, there is no change in the shortform. When the longform has more than four characters the shortform is just the first four characters, unless the fourth character is a vowel. In that case only the first three characters are used.

Note 

There are some commands that do not conform to the truncation rule by design. These will be noted in their respective description pages.

Some examples of how the truncation rule is applied to various commands are shown in table 4-1.

Longform	Shortform
OFF	OFF
DATA	DATA
START	STAR
LONGFORM	LONG
DELAY	DEL
ACCUMULATE	ACC

Table 4-1. Keyword Truncation

Infinity Representation

The representation of infinity is 9.9E+37 for real numbers and 32767 for integers. This is also the value returned when a measurement cannot be

Notation Conventions and Definitions

The following conventions are used in this manual when describing programming rules and examples:

< >	Angular brackets enclose words or characters that are used to symbolize a program code parameter or a bus command.
::=	"is defined as." For example, A ::= B indicates that A can be replaced by B in any statement containing A.
|	"or": indicates a choice of one element from a list. For example, A | B indicates A or B, but not both.
...	An ellipsis (trailing dots) is used to indicate that the preceding element may be repeated one or more times.
[ ]	Square brackets indicate that the enclosed items are optional.
{ }	When several items are enclosed by braces and separated by |s, one, and only one of these elements must be selected.
XXX	Three Xs after an ENTER or OUTPUT statement represent the device address required by your controller.

In addition, the following definition is used:

<NL> ::= Linefeed (ASCII decimal 10).

The Command Tree

The command tree (figure 4-1) shows all commands in the HP 1652B/53B logic analyzers and the relationship of the commands to each other. Parameters are not shown in this figure. The command tree allows you to see what the HP 1652B/53B's parser expects to receive. All legal headers can be created by traversing down the tree, adding keywords until the end of a branch has been reached.

Command Types

As shown in chapter 1's "Header Types" section, there are three types of headers. Each header has a corresponding command type. This section shows how they relate to the command tree.

System Commands. The system commands reside at the top level of the command tree. These commands are always parsable if they occur at the beginning of a program message, or are preceded by a colon. START and STOP are examples of system commands.

Subsystem Commands. Subsystem commands are grouped together under a common node of the tree, such as the MMEMORY commands.

Common Commands. Common commands are independent of the tree, and do not affect the position of the parser within the tree. *CLS and *RST are examples of common commands.

Tree Traversal Rules

Command headers are created by traversing down the command tree. For each group of keywords not separated by a branch, one keyword must be selected. As shown on the tree, branches are always preceded by colons. Do not add spaces around the colons. The following two rules apply to traversing the tree:

A leading colon (the first character of a header) or a <terminator> places the parser at the root of the command tree.

Executing a subsystem command places you in that subsystem (until a leading colon or a <terminator> is found). The parser will stay at the colon above the keyword where the last header terminated. Any command below that point can be sent within the current program message without sending the keywords(s) which appear above them.

Examples The following examples are written using HP BASIC 4.0 on a HP 9000 Series 200/300 Controller. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

The three Xs (XXX) shown in this manual after an ENTER or OUTPUT statement represents the device address required by your controller.

Example 1

```
OUTPUT XXX;":SYSTEM:HEADER ON;LONGFORM ON"
```

In example 1, the colon between SYSTEM and HEADER is necessary since SYSTEM:HEADER is a compound command. The semicolon between the HEADER command and the LONGFORM command is the required < program message unit separator >. The LONGFORM command does not need SYSTEM preceding it, since the SYSTEM:HEADER command sets the parser to the SYSTEM node in the tree.

Example 2

```
OUTPUT XXX;":MMEMORY:INITIALIZE;STORE 'FILE__','FILE DESCRIPTION'"
```

or

```
OUTPUT XXX;":MMEMORY:INITIALIZE"
OUTPUT XXX;":MMEMORY:STORE 'FILE__','FILE DESCRIPTION'"
```

In the first line of example 2, the "subsystem selector" is implied for the STORE command in the compound command. The STORE command must be in the same program message as the INITIALIZE command, since the < program message terminator > will place the parser back at the root of the command tree.

A second way to send these commands is by placing "MMEMORY:" before the STORE command as shown in the fourth line of example 2.

Example 3

```
OUTPUT XXX;":MMEM:CATALOG?;:SYSTEM:PRINT ALL"
```

In example 3, the leading colon before SYSTEM tells the parser to go back to the root of the command tree. The parser can then see the SYSTEM:PRINT command.

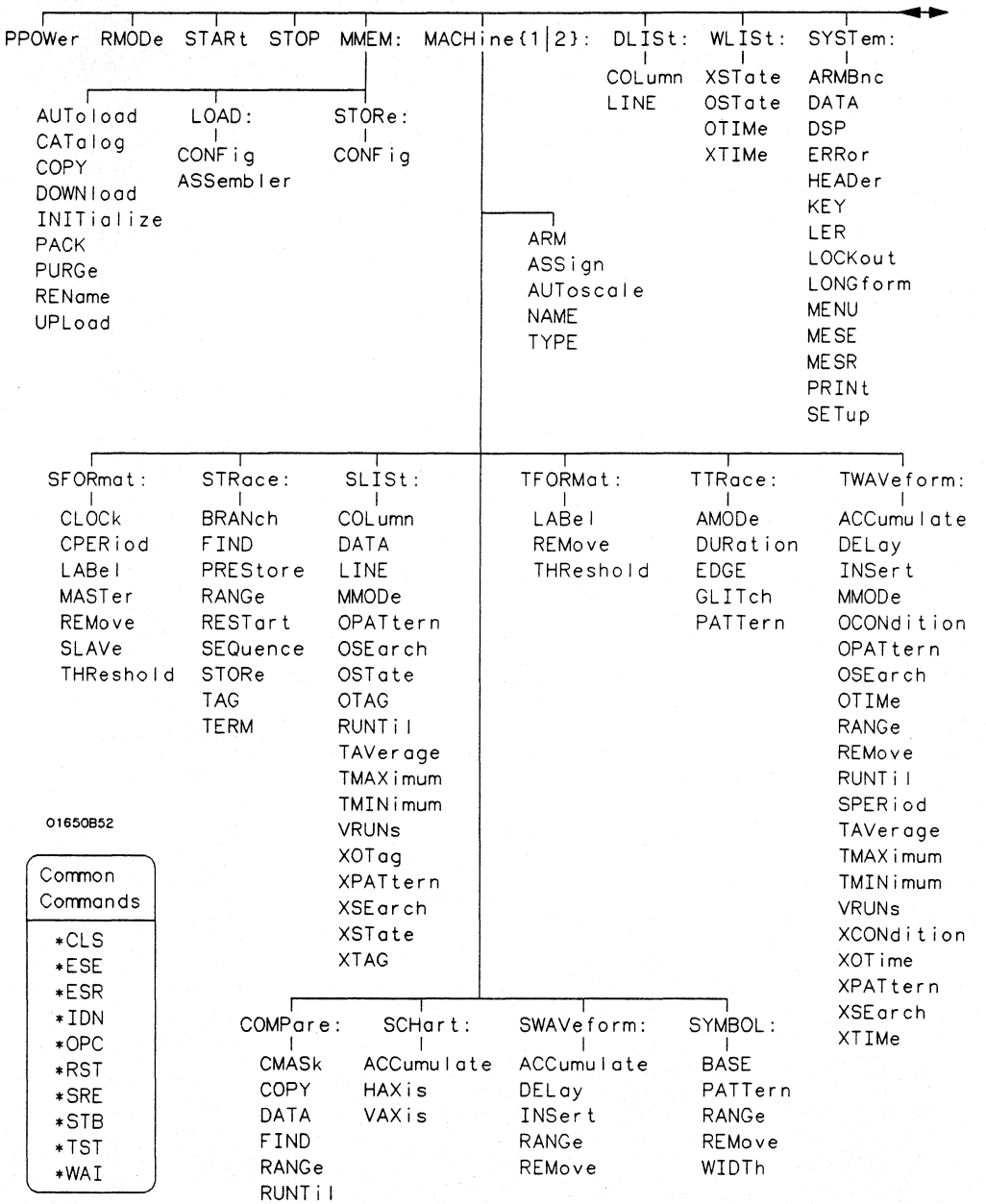


Figure 4-1. HP 1652B/53B Command Tree

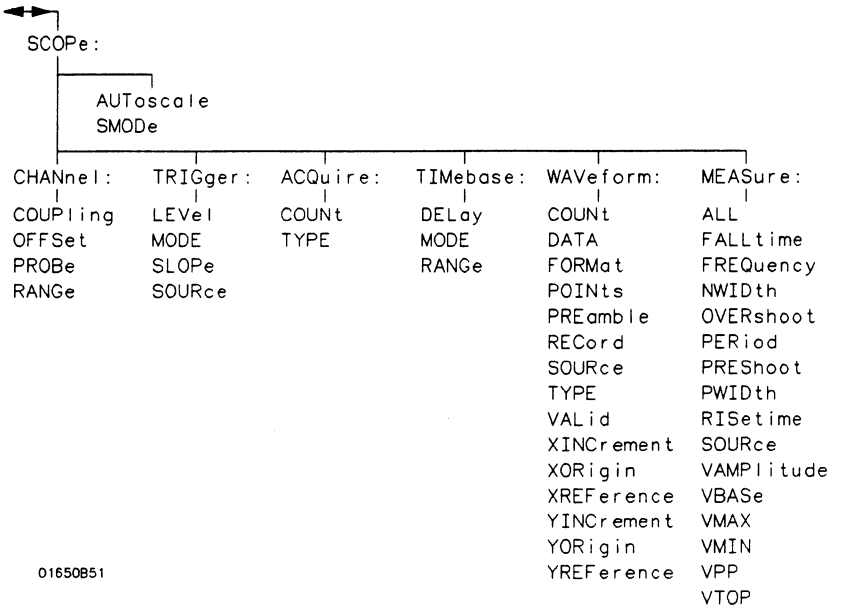


Figure 4-1. HP 1652B/53B Command Tree (continued)

Table 4-2. Alphabetic Command Cross-Reference

Command	Where used	Command	Where used
ACCumulate	SCHart, SWAVeform, TWAVeform	GLITch	TTRace
		HAXis	SCHart
ALL	MEASure	HEADer	System
AMODe	TTRace	INITialize	MMEMory
ARM	MACHine	INSert	SWAVeform, TWAVeform
ARMBnc	System	KEY	System
ASSign	MACHine	LABel	SFORmat, TFORmat
AUToload	MMEMory	LER	System
AUToscale	MACHine, SCOPe	LEVel	TRIGger
BASE	SYMBol	LINE	DLISt, SLISt
BRANch	STRace	LOAD	MMEMory
CATalog	MMEMory	LOCKout	System
CLOCk	SFORmat	LONGform	System
CMASk	COMPress	MASK	SFORmat

Table 4-2. Alphabetic Command Cross-Reference (continued)

Command	Where used	Command	Where used
PATTern	SYMBol, TRace	STORe	MMEMory, STRace
PERiod	MEASure	TAG	STRace
POINts	WAVeform	TAVerage	SLISt, TWAVeform
PPOWer	System	TERM	STRace
PREamble	WAVeform	THReshold	SFORmat, TFORmat
PREShoot	MEASure	TMAXimum	SLISt, TWAVeform
PREStore	STRace	TMINimum	SLISt, TWAVeform
PRINt	System	TYPE	ACQuire, MACHine, WAVeform
PROBe	CHANnel		
PURGe	MMEMory	UPLoad	MMEMory
PWIDth	MEASure	VALid	WAVeform
RANGe	CHANnel, COMPare, STRace, SWAVeform, SYMBol, TIMebase, TWAVeform	VAMPlitude	MEASure
		VAXis	SCHart
		VBASe	MEASure
		VMAX	MEASure
		VMIN	MEASure
RECord	WAVeform	VPP	MEASure
REMove	SFORmat, SWAVeform, Symbol, TFORmat, TWAVeform	VRUNs	SLISt, TWAVeform
		VTOP	MEASure
REName	MMEMory	WIDTh	SYMBol
RESTart	STRace	XCONdition	TWAVeform
RISetime	MEASure	XINCrement	WAVeform
RMODe	System	XORigin	WAVeform
RUNTil	COMPare, SLISt, WAVeform	XOTag	SLISt
		XOTime	TWAVeform
SEQuence	STRace	XPATtern	SLISt, TWAVeform
SETup	System	XREFerence	WAVeform
SLAVe	SFORmat	XSEarch	SLISt, TWAVeform
SLOPe	TRIGger	XSTate	SLISt, WLISt
SMODe	SCOPe	XTAG	SLISt
SOURce	MEASure, TRIGger, WAVeform	XTIMe	TWAVeform, WLISt
		YINCrement	WAVeform
SPERiod	TWAVeform	YORigin	WAVeform
STARt	System	YREFerence	WAVeform
STOP	System		

Command Set Organization

The command set for the HP 1652B/53B logic analyzer is divided into 24 separate groups: common commands, system commands and 22 sets of subsystem commands. Each of the 24 groups of commands is described in the following chapters. Each of the chapters contain a brief description of the subsystem, a set of syntax diagrams for those commands, and finally, the commands for that subsystem in alphabetical order. The commands are shown in the longform and shortform using upper and lowercase letters. As an example AUToload indicates that the longform of the command is AUTOLOAD and the shortform of the command is AUT. Each of the commands contain a description of the command and its arguments, the command syntax, and a programming example.

Subsystems

There are 19 subsystems in this instrument. In the command tree (figure 4-1) they are shown as branches, with the node above showing the name of the subsystem. Only one subsystem may be selected at a time. At power on, the command parser is set to the root of the command tree, and therefore no subsystem is selected. The 22 subsystems in the HP 1652B/53B are:

- SYSTem - controls some basic functions of the instrument.
- MMEMory - provides access to the internal disk drive.
- DLISt - allows access to the dual listing function of two state analyzers.
- WLISt - allows access to the mixed (timing/state) functions.
- MACHine - provides access to analyzer functions and subsystems.
- SFORmat - allows access to the state format functions.
- STRace - allows access to the state trace functions.
- SLISt - allows access to the state listing functions.
- SWAVeform - allows access to the state waveforms functions.
- SCHart - allows access to the state chart functions.
- COMPare - allows access to the compare functions.
- TFORmat - allows access to the timing format functions.
- TTRace - allows access to the timing trace functions.
- TWAVeform - allows access to the timing waveforms functions.
- SYMBol - allows access to the symbol specification functions.
- SCOPe - provides access to oscilloscope functions and subsystems.
- CHANnel - provides access to the vertical axis of the oscilloscope
- TRIGger - allows control of the trigger conditions
- ACQuire - allows control of how the oscilloscope data is acquired.

- TIMebase - allows control of the timebase (horizontal axis) of the oscilloscope.
- WAVeform - allows access to data transfer commands.
- MEASure - allows you to control automated measurements.

Program Examples

The program examples given for each command in the following chapters and appendices were written on an HP 9000 Series 200/300 controller using the HP BASIC 4.0 language. The programs always assume a generic address for the HP 1652/53B of XXX.

In the following examples, special attention should be paid to the ways in which the command and/or query can be sent. Keywords can be sent using either the longform or shortform (if one exists for that word). With the exception of some string parameters, the parser is not case-sensitive. Upper-case (capital) and lower-case (small) letters may be mixed freely. System commands like HEADer and LONGform allow you to dictate what forms the responses take, but have no affect on how you must structure your commands and queries.

The following commands all set Timing Waveform Delay to 100 ms.

- keywords in longform, numbers using the decimal format.

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY .1"
```

- keywords in shortform, numbers using an exponential format.

```
OUTPUT XXX;":MACH1:TWAV:DEL 1E-1"
```

- keywords in shortform using lower-case letters, numbers using a suffix.

```
OUTPUT XXX;":mach1:twav:del 100ms"
```

Note 

In these examples, the colon shown as the first character of the command is optional on the HP 1652B/53B.

The space between DELay and the argument is required.

Common Commands 5

Introduction

The common commands are defined by the IEEE 488.2 standard. These commands will be common to all instruments that comply with this standard.

The common commands control some of the basic instrument functions, such as instrument identification and reset, how status is read and cleared, and how commands and queries are received and processed by the instrument.

Common commands can be received and processed by the HP 1652B/53B whether they are sent over the bus by themselves or as part of a multiple-command string. If an instrument subsystem has been selected and a common command is received by the instrument, the instrument will remain in the selected subsystem. For example, if the instruction

```
":MMEMORY:INITIALIZE;*CLS; STORE 'FILE__','DESCRIPTION'"
```

is received by the instrument, the instrument will initialize the disk and store the file; and clear the status information. This would not be the case if some other type of command were received within the program message. For example, the program message

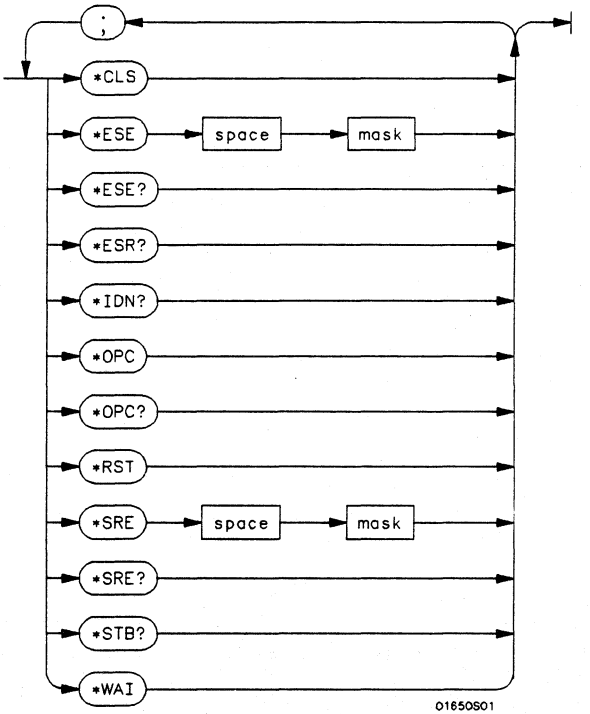
```
":MMEMORY:INITIALIZE;:SYSTEM:HEADERS ON:MMEMORY
:STORE 'FILE__','DESCRIPTION'"
```

would initialize the disk, turn headers on, then store the file. In this example :MMEMORY must be sent again in order to reenter the mmemory subsystem and store the file.

Each status register has an associated status enable (mask) register. By setting the bits in the mask value you can select the status information you wish to use. Any status bits that have not been masked (enabled in the enable register) will not be used to report status summary information to bits in other status registers.

Refer to appendix B, "Status Reporting," for a complete discussion of how to read the status registers and how to use the status information available from this instrument.

Refer to figure 5-1 for the common commands syntax diagram.



mask = *An integer, 0 through 255. This number is the sum of all the bits in the mask corresponding to conditions that are enabled. Refer to the *ESE and *SRE commands for bit definitions in the enable register.*

Figure 5-1. Common Commands Syntax Diagram

*CLS (Clear Status) command

The *CLS common command clears the status data structures, including the device defined error queue. If the *CLS command immediately follows a <terminator>, the output queue and the MAV (Message Available) bit will be cleared.

Command Syntax: *CLS

Example:

```
OUTPUT XXX;"*CLS"
```

Note Refer to appendix B, "Status Reporting," for a complete discussion of status.

*ESE (Event Status Enable) command/query

The *ESE command sets the Standard Event Status Enable Register bits. The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A one in the Standard Event Status Enable Register will enable the corresponding bit in the Standard Event Status Register. A zero will disable the bit. Refer to table 4-1 for information about the Standard Event Status Enable Register bits, bit weights, and what each bit masks.

The *ESE query returns the current contents of the enable register.

Note Refer to appendix B, "Status Reporting," for a complete discussion of status.

Command Syntax: *ESE <mask>

where:

<mask> ::= integer from 0 to 255

Example:

```
OUTPUT XXX;"*ESE 32"
```

In this example, the *ESE 32 command will enable CME (Command Error), bit 5 of the Standard Event Status Enable Register. Therefore, when a command error occurs, the event summary bit (ESB) in the Status Byte Register will also be set.

Query Syntax: *ESE?

Returned Format: <mask><NL>

Example:
```
10 DIM Event$[100]
20 OUTPUT XXX;"*ESE?"
30 ENTER XXX;Event$
40 PRINT Event$
50 END
```

Table 5-1. Standard Event Status Enable Register

Bit	Weight	Enables
7	128	PON - Power On
6	64	URQ - User Request
5	32	CME - Command Error
4	16	EXE - Execution Error
3	8	DDE - Device Dependent Error
2	4	QYE - Query Error
1	2	RQC - Request Control
0	1	OPC - Operation Complete

High - enables the ESR bit

*ESR (Event Status Register) query

The *ESR query returns the contents of the Standard Event Status Register. Reading the register clears the Standard Event Status Register.

Note The bits in this register must be set by sending the *ESE command before sending the *ESR query (see "*ESE command/query" on page 5-4).

Query Syntax: *ESR?

Returned Format: <status> <NL>

where:

<status> ::= integer from 0 to 255

Example:

```
10 DIM Esr_event$[100]
20 OUTPUT XXX;"*ESR?"
30 ENTER XXX;Esr_event$
40 PRINT Esr_event$
50 END
```

With the example, if a command error has occurred the variable "Esr_event" will have bit 5 (the CME bit) set.

Table 4-2 shows the Standard Event Status Register. The table shows each bit in the Standard Event Status Register, and the bit weight. When you read Standard Event Status Register, the value returned is the total bit weights of all bits that are high at the time you read the byte.

Table 5-2. The Standard Event Status Register.

BIT	BIT WEIGHT	BIT NAME	CONDITION
7	128	PON	0 = Register read - not in power up mode 1 = Power up
6	64	URQ	0 = user request - not used - always zero
5	32	CME	0 = no command errors 1 = a command error has been detected
4	16	EXE	0 = no execution errors 1 = an execution error has been detected
3	8	DDE	0 = no device dependent errors 1 = a device dependent error has been detected
2	4	QYE	0 = no query errors 1 = a query error has been detected
1	2	RQC	0 = request control - NOT used - always 0
0	1	OPC	0= operation is not complete 1 = operation is complete

0 = False = Low
1 = True = High

*IDN (Identification Number) query

The *IDN? query allows the instrument to identify itself. It returns the string:

"HEWLETT-PACKARD,1652B,0,REV <revision code>"

An *IDN? query must be the last query in a message. Any queries after the *IDN? in the program message will be ignored.

Query Syntax: *IDN?

Returned Format: HEWLETT-PACKARD,1652B,0,REV <revision code>

where:

<revision code> ::= four-digit code representing ROM revision

Example:

```
10 DIM Id$[100]
20 OUTPUT XXX;"*IDN?"
30 ENTER XXX;Id$
40 PRINT Id$
50 END
```

*OPC (Operation Complete) command/query

The *OPC command will cause the instrument to set the operation complete bit in the Standard Event Status Register when all pending device operations have finished. The commands which affect this bit are the Overlapped Commands. An Overlapped Command is a command that allows execution of subsequent commands while the device operations initiated by the Overlapped Command are still in progress. The overlapped commands for the HP 1652B/53B are:

STARt
STOP
AUToscale

The *OPC query places an ASCII "1" in the output queue when all pending device operations have been completed.

Command Syntax: *OPC

Example:
```
OUTPUT XXX;"*OPC"
```

Query Syntax: *OPC?

Returned Format: 1<NL>

Example:
```
10 DIM Status$[100]
20 OUTPUT XXX;"*OPC?"
30 ENTER XXX;Status$
40 PRINT Status$
50 END
```

*RST (Reset) command

The *RST command (488.2) sets the HP 1652B/53B to the power-up default settings as if no autoload file was present.

Command Syntax: *RST

Example:
```
OUTPUT XXX;"*RST"
```

*SRE (Service Request Enable) command/query

The *SRE command sets the Service Request Enable Register bits. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register will enable the corresponding bit in the Status Byte Register. A zero will disable the bit. Refer to table 5-3 for the bits in the Service Request Enable Register and what they mask.

The *SRE query returns the current value.

Note Refer to appendix B, "Status Reporting," for a complete discussion of status.

Command Syntax: *SRE <mask>

where:

<mask> ::= integer from 0 to 255

Example:

```
OUTPUT XXX;"*SRE 16"
```

This example forces the MAV bit high (see table 5-3).

Query Syntax: *SRE?

Returned Format: <mask><NL>

where:

<mask> ::= sum of all bits that are set - 0 through 255

Example:

```
10 DIM Sre_value$[100]
20 OUTPUT XXX;"*SRE?"
30 ENTER XXX;Sre_value$
40 PRINT Sre_value$
50 END
```

Table 5-3. HP 1652B/53B Service Request Enable Register

Bit	Weight	Enables
15-8		not used
7	128	not used
6	64	MSS - Master Summary Status
5	32	ESB - Event Status
4	16	MAV - Message Available
3	8	not used
2	4	not used
1	2	LCL - Local
0	1	MSB - Module Summary

*STB (Status Byte) query

The *STB query returns the current value of the instrument's status byte. The MSS (Master Summary Status) bit and not RQS (Request Service) bit is reported on bit 6. The MSS indicates whether or not the device has at least one reason for requesting service. Refer to table 5-4 for the meaning of the bits in the status byte.

Note Refer to appendix B, "Status Reporting," for a complete discussion of status.

Query Syntax: *STB?

Returned Format: <value><NL>

where:

<value> ::= integer from 0 to 255

Example:

```
10 DIM Stb_value$[100]
20 OUTPUT XXX;"*STB?"
30 ENTER XXX;Stb_value$
40 PRINT Stb_value$
50 END
```

Table 5-4. The Status Byte Register

BIT	BIT WEIGHT	BIT NAME	CONDITION
7	128	---	0 = not used
6	64	MSS	0 = instrument has no reason for service 1 = instrument is requesting service
5	32	ESB	0 = no event status conditions have occurred 1 = an enabled event status condition has occured
4	16	MAV	0 = no output messages are ready 1 = an output message is ready
3	8		not used
2	4	---	not used
1	2	LCL	0 = a remote-to-local transition has not occurred 1 = a remote-to-local transition has occurred
0	1	MSB	0 = HP 1652B/1653B has activity to report 1 = no activity to report

0 = False = Low
1 = True = High

*WAI (Wait) command

The *WAI command causes the device to wait until the completion of all overlapped commands before executing any further commands or queries. An overlapped command is a command that allows execution of subsequent commands while the device operations initiated by the overlapped command are still in progress. The overlapped commands for the HP 1652B/53B are:

STARt
STOP
AUToscale

Command Syntax: *WAI

Example:

```
OUTPUT XXX;"*WAI"
```

System Commands 6

Introduction

System commands control the basic operation of the instrument including formatting query responses and enabling reading and writing to the advisory line of the instrument's display. They can be called at anytime. The HP 1652B/53B System commands are:

- ARMBnc
- DATA
- DSP (display)
- ERRor
- HEADer
- KEY
- LER (Local Event Register)
- LOCKout
- LONGform
- MENU
- MESE
- MESR
- PRINt
- SETup

In addition to the system commands, there is are three run control commands and a preprocessor power supply condition query. These commands are:

- PPOWer
- RMODe
- STARt
- STOP

The run control commands can be called at anytime and also control the basic operation of the logic analyzer. These commands are at the same level in the command tree as SYSTem; therefore they are not preceded by the :SYSTem header.

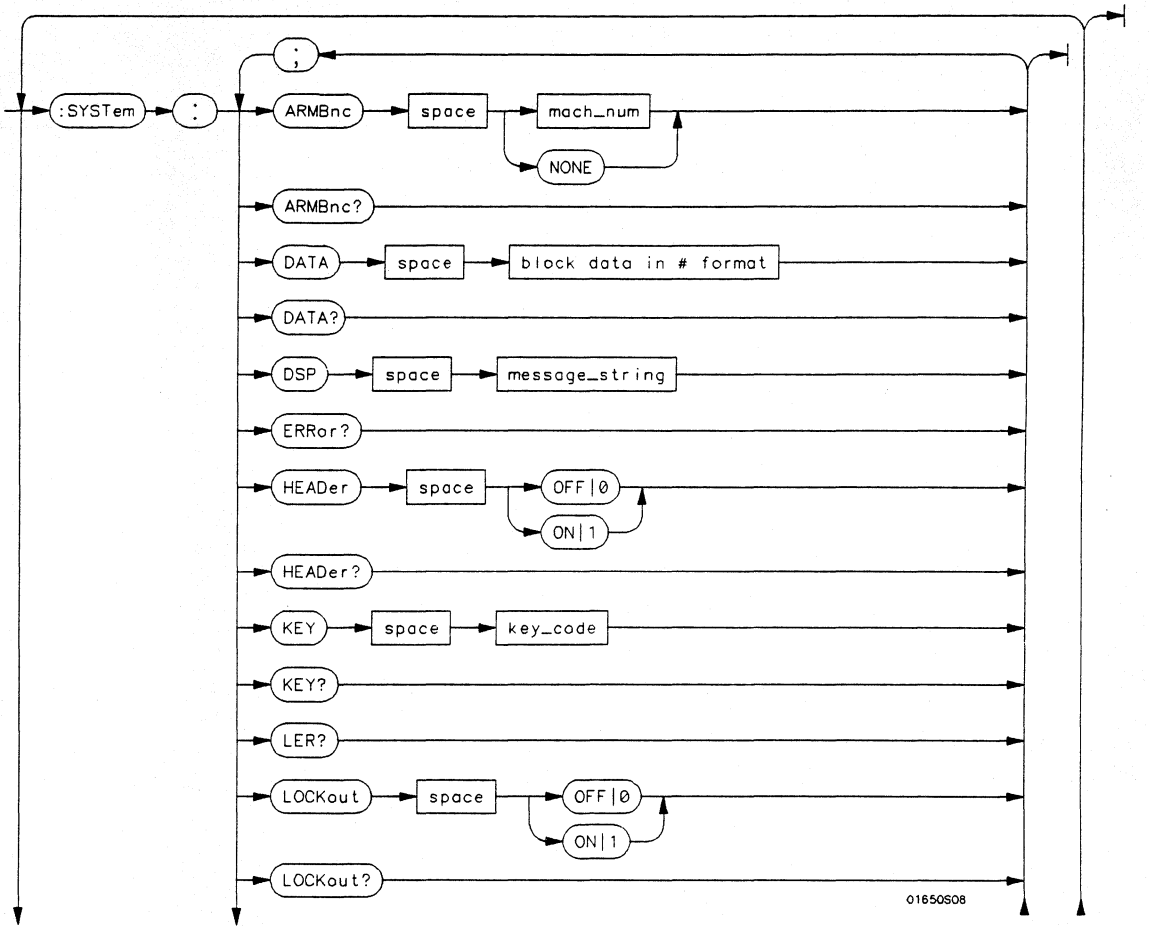
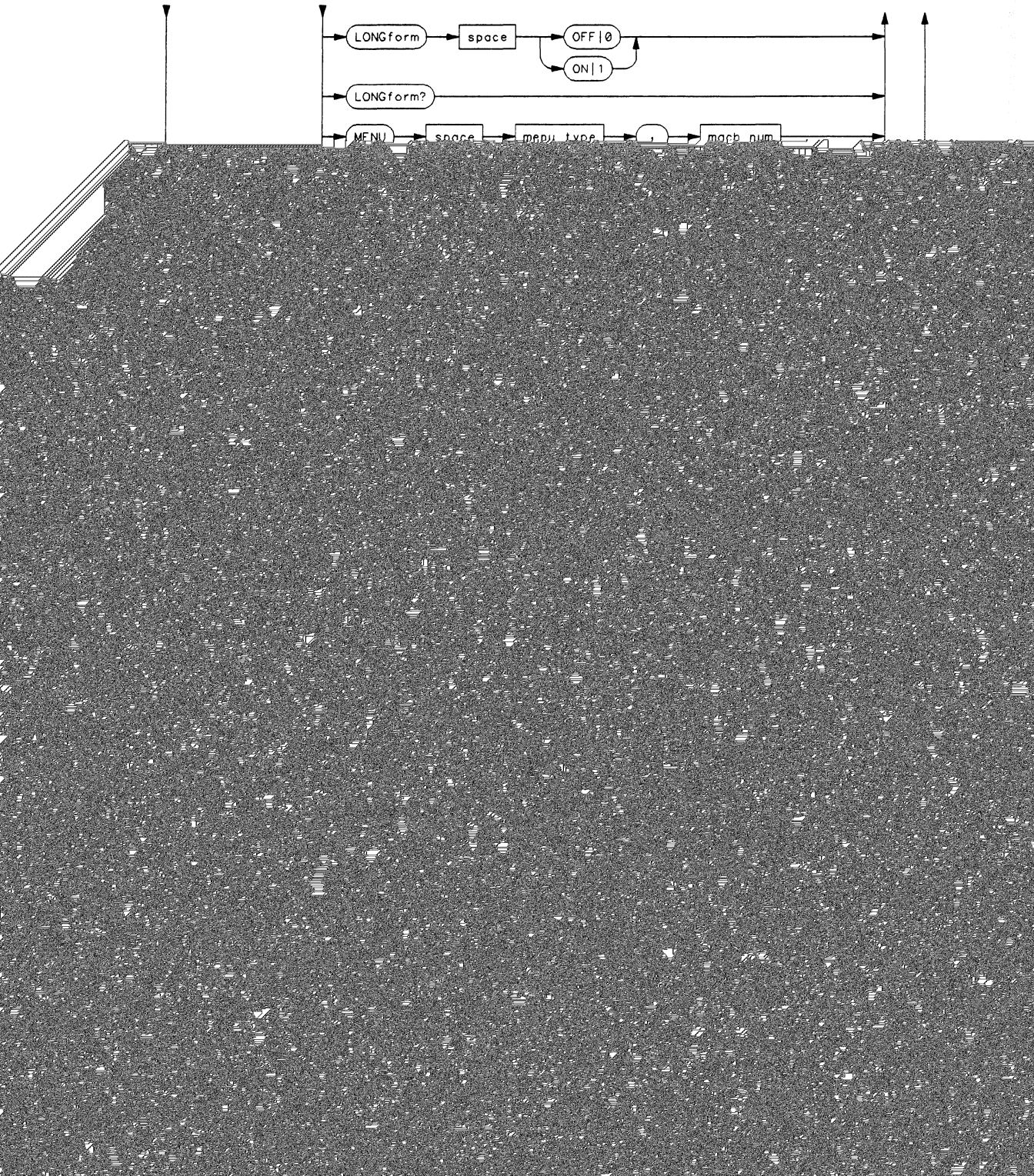


Figure 6-1. System Commands Syntax Diagram


LONGform
space
OFF|0
ON|1
LONGform?
MENU
space
menu type
,
mach num

ARMBnc

command/query

The ARMBnc command selects the source that will generate the arm out signal that will appear on the rear panel BNC labelled External Trigger Out.

The ARMBnc query returns the source currently selected.

Command Syntax: :SYSTem:ARMBnc {MACHine{1|2}|SCOPe | NONE}

Example:
```
OUTPUT XXX;":SYSTEM:ARMBNC MACHINE1"
```

Query Syntax: :SYSTem:ARMBnc?

Returned Format: [:SYSTem:ARMBnc] {MACHine{1|2}|SCOPe |NONE}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":ARMBNC?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

DATA

command/query

The DATA command allows you to send and receive acquired data to and from a controller in block form. This helps saving block data for:

- Re-loading to the logic analyzer
- Processing data later
- Processing data in the controller.

The format and length of block data depends on the instruction being used and the configuration of the instrument. This section describes each part of the block data as it will appear when used by the DATA instruction. The beginning byte number, the length in bytes, and a short description is given for each part of the block data. This is intended to be used primarily for processing of data in the controller.

Note

Do not change the block data in the controller if you intend to send the block data back into the logic analyzer for later processing. Changes made to the block data in the controller could have unpredictable results when sent back to the logic analyer.

The SYSTem:DATA query returns the block data.

Note

The data sent by the SYSTem:DATA query reflects the configuration of the machines when the last run was performed. Any changes made since then through either front-panel operations or programming commands do not affect the stored configuration.

For the DATA instruction, block data consists of either 14506 bytes containing logic analyzer only information or 26794 bytes containing both logic analyzer and oscilloscope information. This information is captured by the acquisition systems. The information for the logic analyzer will be in one of four formats depending on the type of data captured. The logic analyzer format is described in the "Acquisition Data Description" section in "Logic Analyzer Block Data." The oscilloscope format is described in the "Acquisition Data Description" section in "Oscilloscope Block Data." Since no parameter checking is performed, out-of-range values could cause instrument lockup; therefore, care should be taken when transferring the data string into the HP 1652B/53B.

The <block data> parameter can be broken down into a <block length specifier> and a variable number of <section>s.

The <block length specifier> always takes the form #8DDDDDDDD. Each D represents a digit (ASCII characters "0" through "9"). The value of the eight digits represents the total length of the block (all sections). For example, if the total length of the block is 14522 bytes, the block length specifier would be "#800014522".

Each <section> consists of a <section header> and <section data>. The <section data> format varies for each section and may be any length. For this instruction, the <section data> section is composed of a data preamble section and an acquisition data section.

Command Syntax: :SYSTem:DATA <block data>

Example:
```
OUTPUT XXX;":SYSTEM:DATA" <block data>
```

where:

<block data>	::= <block length specifier> <section>...
<block length specifier>	::= #8<length>
<length>	::= the total length of all sections in byte format (must be represented with 8 digits)
<section>	::= <section header> <section data>
<section header>	::= 16 bytes, described in the following "Section Header" sections
<section data>	::= format depends on the type of data

Note 

The total length of a section is 16 (for the section header) plus the length of the section data. So when calculating the value for <length>, don't forget to include the length of the section headers.

Query Syntax: :SYSTem:DATA?

Returned Format: [:SYSTem:DATA] <block data><NL>

HP-IB Example:
```
10 DIM Num$[2], Block$[32000]     !  allocate enough memory for block data
20 OUTPUT XXX;":SYSTEM:HEAD OFF"
30 OUTPUT XXX;":SYSTEM:DATA?"     ! send data query
40 ENTER XXX USING "#,2A";Num$    !read in #8
50 ENTER XXX USING "#,8D";Blocklength! read in block length
60 ENTER XXX USING "-K";Block$    ! read in data
70 END
```

Logic Analyzer Block Data

The logic analyzer block data is described in the following sections. The oscilloscope block data is appended at the end of the logic analyzer block data when the oscilloscope is on and has acquired and stored waveform data. The oscilloscope block data is described in "Oscilloscope Block Data" later in this section.

Section Header Description

The section header uses bytes 1 through 16 (this manual begins counting at 1; there is no byte 0). The **16 bytes** of the section header are as follows:

1 **10 bytes** - section name, such as "DATA " (six trailing spaces)

11 **1 byte** - reserved

12 **1 bytes** - module ID (31 for HP 1652B/53B)

13 **4 bytes** - length (14506 for the logic analyzer only and 26794 for both the logic analyzer and oscilloscope).

Section Data

For the SYSTem:DATA command, the <section data> parameter consists of two parts: the data preamble and the acquisition data. These are described in the following two sections.

Data Preamble Description

The block data is organized as 160 bytes of preamble information, followed by 1024 14-byte groups of information, followed by 10 reserved bytes. The preamble gives information for each analyzer describing the amount and type of data captured, where the trace point occurred in the data, which pods are assigned to which analyzer, and other information.

Each 14-byte group is made up of two bytes (16 bits) of status for Analyzer 1, two bytes of status for Analyzer 2, then five sets of two bytes of information for each of the five 16-bit pods of the HP 1652B. In the HP 1653B, the status and format for the sets of bytes are the same, but the data in not valid on pods 3, 4, and 5.

Note

One analyzer's information is independent of the other analyzer's information. In other words, on any given line, one analyzer may contain data information for a timing machine, while the other analyzer may contain count information for a state machine with time tags enabled. The status bytes for each analyzer describe what the information for that line contains. Therefore, when describing the different formats that data may contain below, keep in mind that this format pertains only to those pods that are assigned to the analyzer of the specified type. The other analyzer's data is TOTALLY independent and conforms to its own format.

The preamble (bytes 17 through 176) consists of the following **160 bytes**:

17 **2 bytes** - Instrument ID (always 1652 for HP 1652B and HP 1653B)

19 **2 bytes** - Revision Code

Note

The values stored in the preamble represent the captured data currently stored in this structure and not what the current configuration of the analyzer is. For example, the mode of the data (bytes 21 and 99) may be STATE with tagging, while the current setup of the analyzer is TIMING.

The next **78 bytes** are for Analyzer 1 Data Information.

21 **1 byte** - Machine data mode, one of the following values:

0 = off
1 = state data (with either time or state tags)
2 = state data (without tags)
3 = glitch timing data
4 = transitional timing data

22 **1 byte** - List of pods in this analyzer, where a 1 indicates that the corresponding pod is assigned to this analyzer.

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
unused	unused	Pod 1	Pod 2	Pod 3	Pod 4	Pod 5	unused

23 **1 byte** - Master chip in this analyzer - When several chips are grouped together in a single analyzer, one chip is designated as a master chip. This byte identifies the master chip. A value of 4 represents POD 1, 3 for POD 2, 2 for POD 3, 1 for POD 4, and 0 for POD 5.

24 **1 byte** - Reserved

25 **10 bytes** - Number of rows of valid data for this analyzer - Indicates the number of rows of valid data for each of the five pods. Two bytes are used to store each pod value, with the first 2 bytes used to hold POD 5 value, the next 2 for POD 4 value, and so on.

35 **1 byte** - Trace point seen in this analyzer - Was a trace point seen (value = 1) or forced (value = 0)

36 **1 byte** - Reserved

37 **10 bytes** - Trace point location for this analyzer - Indicates the row number in which the trace point was found for each of the five pods. Two bytes are used to store each pod value, with the first 2 bytes used to hold POD 5 value, the next 2 for POD 4 value, and so on.

47 **4 bytes** - Time from arm to trigger for this analyzer - The number of 40 ns ticks that have taken place from the arm of this machine to the trigger of this machine. A value of -1 (all 32 bits set to 1) indicates counter overflow.

51 **1 byte** - Armer of this analyzer - Indicates what armed this analyzer (1 = RUN, 2 = BNC, 3 = other analyzer, 4 = SCOPE)

52 **1 byte** - Devices armed by this analyzer - Bitmap of devices armed by this machine

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
unused	unused	unused	unused	SCOPE	BNC out	Mach. 2	Mach. 1

A 1 in a given bit position implies that this analyzer arms that device, while a 0 means the device is not armed by this analyzer.

53 **4 bytes** - Sample period for this analyzer (timing only) - Sample period at which data was acquired. Value represents the number of nanoseconds between samples.

57 **4 bytes** - Delay for this analyzer (timing only) - Delay at which data was acquired. Value represents the amount of delay in nanoseconds.

61 **1 byte** - Time tags on (state with tagging only) - In state tagging mode, was the data captured with time tags (value = 1) or state tags (value = 0).

62 **1 byte** - Reserved

63 **5 bytes** - Demultiplexing (state only) - For each of the five pods (first byte is POD 5, fifth byte is POD 1) in a state machine, describes multiplexing of each of the five pods. (0 = NO DEMUX, 1 = TRUE DEMUX, 2 = MIXED CLOCKS).

68 **1 byte** - Reserved

69 **20 bytes** - Trace point adjustment for pods - Each pod uses 4 bytes to show the number of nanoseconds that are to be subtracted from the trace point described above to get the actual trace point value. The first 4 bytes are for Pod 5, the next four are for Pod 4, and so on.

89 **10 bytes** - Reserved

The next **78 bytes** are for Analyzer 2 Data Information. They are organized in the same manner as Analyzer 1 above, but they occupy bytes 99 through 176

Acquisition Data Description

The acquisition data section consists of 14336 bytes (1024 14-byte groups), appearing in bytes 177 through 14512. The last ten bytes (14513 through 14522) are reserved. The data contained in the data section will appear in one of four forms depending on the mode in which it was acquired (as indicated in byte 21 for machine 1 and byte 99 for machine 2). The four modes are:

- State Data (without tags)
- State Data (with either time or state tags)
- Glitch Timing Data
- Transitional Timing Data

The following four sections describe the four data modes that may be encountered. Each section describes the Status bytes (shown under the Machine 1 and Machine 2 headings), and the Information bytes (shown under the Pod 5 through Pod 1 headings).

State Data (without tags)

Status Bytes. In normal state mode, only the least significant bit (bit 1) is used. When bit 1 is set, this means that there has been a sequence level transition.

Information Bytes. In state acquisition with no tags, data is obtained from the target system with each clock and checked with the trace specification. If the state matches this specification, the data is stored, and is placed into the memory.

	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	Data	Data	Data	Data	Data
205	Status	Status	Data	Data	Data	Data	Data
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14499	Status	Status	Data	Data	Data	Data	Data

*The headings are not a part of the returned data.

State Data (with either time or state tags)

Status Bytes. In state tagging mode, the tags indicate whether a given row of the data is a data line, a count (tag) line, or a prestore line.

Bit 2 is the Data vs. Count bit. Bit 3 is the Prestore vs. Tag bit. The two bits together show what the corresponding Information bytes represent.

Bit 3	Bit 2	Information byte represents:
0	0	Acquisition Data
0	1	Count
1	0	Prestore Data
1	1	Invalid

If Bit 2 is clear, the information contains either actual acquisition data as obtained from the target system (if Bit 3 is clear), or prestore data (if Bit 3 is set). If Bit 2 is set and Bit 3 is clear, this row's bytes for the pods assigned to this machine contain tags. If Bit 2 and Bit 3 are set, the corresponding Information bytes are invalid and should be ignored. Bit 1 is used only when Bit 2 is clear. Whenever there has been a sequence level transition Bit 1 will be set, and otherwise will be clear.

Information Bytes. In the State acquisition mode with tags, data is obtained from the target system with each clock and checked with the trace specification. If the state does not match the trace specification, it is checked against the prestore qualifier. If it matches the prestore qualifier, then it is placed in the prestore buffer. If the state does not match either the sequencer qualifier or the prestore qualifier, it is discarded.

The type of information in the bytes labeled Data depends on the Prestore vs. Tags bit. When the Data bytes are used for prestore information, the following Count bytes (in the same column) should be ignored. When the Data bytes are used for tags, the Count bytes are formatted as floating-point numbers in the following fashion:

bits 16 through 12	bits 11 through 1
EEEEE	MMMMMMMMMMM

The five most-significant bits (EEEEE) store the exponent, and the eleven least-significant bits (MMMMMMMMMMM) store the mantissa. The actual value for Count is given by the equation:

$$\text{Count} = (2048 + \text{mantissa}) \times 2^{\text{exponent}} - 2048$$

Since the counts are relative counts from one state to the one previous, the count for the first state in the data structure is invalid.

If time tagging is on, the count value represents the number of 40 nanosecond ticks that have elapsed between the two stored states. In the case of state tagging, the count represents the number of qualified states that were encountered between the stored states.

If a state matches the sequencer qualifiers, the prestore buffer is checked. If there are any states in the prestore buffer at this time, these prestore states are first placed in memory, along with a dummy count row. After this check, the qualified state is placed in memory, followed by the count row which specified how many states (or 40 ns ticks) have elapsed since the last stored state. If this is the first stored state in memory, then the count information that is stored should be discarded.

	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	⊗	⊗	⊗	⊗	⊗
205	Status	Status	Data	Data	Data	Data	Data
219	Status	Status	Count	Count	Count	Count	Count
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14485	Status	Status	Data	Data	Data	Data	Data
14499	Status	Status	Count	Count	Count	Count	Count

*The headings are not a part of the returned data.

⊗ = Invalid data

Glitch Timing Data

Status Bytes. In glitch timing mode, the status bytes indicate whether a given row in the data contains actual acquisition data information or glitch information.

Bit 1 is the Data vs. Glitch bit. If Bit 1 is set, this row of information contains glitch information. If Bit 1 is clear, then this row contains actual acquisition data as obtained from the target system.

Information Bytes. In the Glitch timing mode, the target system is sampled at every sample period. The data is then stored in memory and the glitch detectors are checked. If a glitch has been detected between the previous sample and the current sample, the corresponding glitch bits are set. The glitch information is then stored. If this is the first stored sample in memory, then the glitch information stored should be discarded.

	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	⊗	⊗	⊗	⊗	⊗
205	Status	Status	Data	Data	Data	Data	Data
219	Status	Status	Glitch	Glitch	Glitch	Glitch	Glitch
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14485	Status	Status	Data	Data	Data	Data	Data
14499	Status	Status	Glitch	Glitch	Glitch	Glitch	Glitch

*The headings are not a part of the returned data.

⊗ = Invalid data

Transitional Timing Data

Status Bytes. In transitional timing mode, the status bytes indicate whether a given row in the data contains acquisition information or transition count information.

bits 10-9	bits 8-7	bits 6-5	bits 4-3	bits 2-1
Pod 5	Pod 4	Pod 3	Pod 2	Pod 1

Each pod uses two bits to show what is being represented in the corresponding Information bytes. Bits 10, 8, 6, 4 and 2 are set when the appropiate pod's Information bytes represent acquisition data. When that bit is clear, the next bit shows if the Information bytes represent the first word of a count. Together there are three possible combinations:

10 - This pod's Information bytes contain acquisition data as obtained from the target system.
01 - This pod's Information bytes contain the first word of a count.
00 - This pod's Information bytes contain part of a count other than the first word.

Information Bytes. In the Transitional timing mode the logic analyzer performs the following steps to obtain the information bytes:

1. Four samples of data are taken at 10 nanosecond intervals. The data is stored and the value of the last sample is retained.

2. Four more samples of data are taken. If any of these four samples differ from the last sample of the step 1, then these four samples are stored and the last value is once again retained.

3. If all four samples of step 2 are the same as the last sample taken in step 1, then no data is stored. Instead, a counter is incremented. This process will continue until a group of four samples is found which differs from the retained sample. At this time, the count will be stored in the memory, the counters reset, the current data stored, and the last sample of the four once again retained for comparison.

Note 

The stored count indicates the number of 40 ns intervals that have elapsed between the old data and the new data.

The rows of the acquisition data may, therefore, be either four rows of data followed by four more rows of data, or four rows of data followed by four rows of count. Rows of count will always be followed by four rows of data except for the last row, which may be either data or count.

Note 

This process is performed on a pod-by-pod basis. The individual status bits will indicate what each pod is doing.

The following table is just an example. The meaning of the Information bytes (Data or Count) depends upon the corresponding Status bytes.

Example:	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	Data	Data	Data	Data	Data
205	Status	Status	Data	Data	Data	Data	Data
219	Status	Status	Data	Data	Data	Data	Data
233	Status	Status	Data	Count	Count	Data	Data
247	Status	Status	Data	Count	Count	Data	Data
261	Status	Status	Data	Count	Count	Data	Data
275	Status	Status	Data	Count	Count	Data	Data
289	Status	Status	Count	Data	Data	Count	Data
303	Status	Status	Count	Data	Data	Count	Data
317	Status	Status	Count	Data	Data	Count	Data
331	Status	Status	Count	Data	Data	Count	Data
345	Status	Status	Data	Data	Count	Data	Data
359	Status	Status	Data	Data	Count	Data	Data
373	Status	Status	Data	Data	Count	Data	Data
387	Status	Status	Data	Data	Count	Data	Data
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14457	Status	Status	Data	Data	Data	Data	Data
14471	Status	Status	Data	Data	Data	Data	Data
14485	Status	Status	Data	Data	Data	Data	Data
14499	Status	Status	Data	Data	Data	Data	Data

*The headings are not a part of the returned data.

Oscilloscope Block Data

The oscilloscope block data is described in the following sections. This data is appended to the logic analyzer block data and is present only when the oscilloscope is on and waveform data has been acquired and stored.

The oscilloscope data contains both a section header and section data similar to the logic analyzer for both of its sections. The oscilloscope block data sections are Oscilloscope Data and Oscilloscope Display Data.

- Oscilloscope Data - the raw data captured on the last acquisition.
- Oscilloscope Display Data - the segment of data displayed after each acquisition.

The oscilloscope data and oscilloscope display data sections are sent only when the oscilloscope is on and there is waveform data stored in the oscilloscope memory .

Oscilloscope Data Section

The Oscilloscope Data section contains the raw data the oscilloscope acquired on the last acquisition.

Section Header Description

The oscilloscope data <section header> used bytes 14523 through 14539. The **16 bytes** of the section header are as follows:

14523 **10 bytes** - Section name, "SCOPEDAT " (two trailing spaces)

14533 **1 byte** - Reserved (always 0)

14534 **1 byte** - Unused

14535 **4 bytes** - Length of oscilloscope data

Section Data

The oscilloscope raw data <section data> contains the initially acquired data. Each data unit is contained in a byte. The lower six bits contain the data, while the upper two bits are not used and as a result, each data unit can represent a value from 0 to 63. The total number of bytes is this section is 4096 with the first 2048 bytes for channel 1 and the remaining 2048 bytes for channel 2.

14539 **2048 bytes** - raw oscilloscope data for channel 1.

16587 **2048 bytes** - raw oscilloscope data for channel 1.

Oscilloscope Display Data Section

The display data section <section data> contains the initial data displayed after an acquisition. Each data unit is represented by a 16 bit value which is generated by taking the raw oscilloscope data and shifting it the the left by 8 bits.

Note 

Changing the seconds-per-division after the oscilloscope has stopped will change the data displayed on the screen but it will not change the display data in this section.

18635 **4096 bytes** - Displayed oscilloscope data for channel 1

22731 **4096 bytes** - Displayed oscilloscope data for channel 2

DSP (Display) command

The DSP command writes the specified quoted string to a device dependent portion of the instrument display.

Command Syntax: :SYSTem:DSP <string>

where:

<string> ::= string of up to 60 alphanumeric characters

Examples:

```
OUTPUT XXX;":SYSTEM:DSP 'The message goes here'"
```

ERRor query

The ERRor query returns the oldest error number from the error queue. A complete list of error numbers for the HP 1652B/53B is shown in appendix C, "Error Messages." If no errors are present in the error queue, a zero is returned.

Query Syntax: :SYSTem:ERRor?

Returned Format: [:SYSTem:ERRor] <error number> <NL>

Example:

```
10 OUTPUT XXX;":SYSTEM:ERROR?"
20 ENTER XXX;Err_num
30 PRINT Err_num
40 END
```

HEADer command/query

The HEADER command tells the instrument whether or not to output a header for query responses. When HEADer is set to ON, query responses will include the command header.

The HEADer query returns the current state of the HEADer command.

Command Syntax: :SYSTem:HEADer {{ON|1}|{OFF|0}}

Example:
```
OUTPUT XXX;":SYSTEM:HEADER ON"
```

Query Command: :SYSTem:HEADer?

Returned Format: [:SYSTem:HEADer] {1|0}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":SYSTEM:HEADER?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

KEY

command/query

The KEY command allows you to simulate pressing a specified front-panel key. Key commands may be sent over the bus in any order that is legal from the front panel. Be sure the instrument is in a desired setup before executing the KEY command. Key codes range from 0 to 36 with 99 representing no key (returned at power-up). See table 6-1 for key codes.

Note 

The external KEY buffer is only two keys deep; therefore, attempting to send KEY commands too rapidly will cause a KEY buffer overflow error to be displayed on the HP 1652B/53B screen.

The KEY query returns the key code for the last front- panel key pressed or the last simulated key press over the bus.

Command Syntax: :SYSTem:KEY <key_code>

where:

<key_code> ::= integer from 0 to 36

Example:

```
OUTPUT XXX;":SYSTEM:KEY 24"
```

KEY

Query Syntax: :SYSTem:KEY?

Returned Format: [:SYSTem:KEY] <value><NL>

LER (LCL Event Register) query

The LER query allows the LCL (local) Event Register to be read. After the LCL Event Register is read, it is cleared. A one indicates a remote-to-local transition has taken place. A zero indicates a remote-to-local transition has not taken place.

Query Syntax: :SYSTem:LER?

Returned Format: [:SYSTem:LER] {0|1}<NL>

Example:

```
10 DIM Event$[100]
20 OUTPUT XXX;":SYSTEM:LER?"
30 ENTER XXX;Event$
40 PRINT Event$
50 END
```

LOCKout command/query

The LOCKout command locks out or restores front-panel operation. When this function is on, all controls (except the power switch) are entirely locked out.

The LOCKout query returns the current status of the LOCKout command.

Command Syntax: :SYSTem:LOCKout {{ON|1}|{OFF|0}}

Example:
```
OUTPUT XXX;":SYSTEM:LOCKOUT ON"
```

Query Syntax: :SYSTem:LOCKout?

Returned Format: [:SYSTem:LOCKout] {0|1}<NL>

Example:
```
10 DIM Status$[100]
20 OUTPUT XXX;":SYSTEM:LOCKOUT?"
30 ENTER XXX;Status$
40 PRINT Status$
50 END
```

LONGform

command/query

The LONGform command sets the longform variable which tells the instrument how to format query responses. If the LONGform command is set to OFF, command headers and alpha arguments are sent from the instrument in the abbreviated form. If the LONGform command is set to ON, the whole word will be sent to the controller.

This command has no affect on the input data messages to the instrument. Headers and arguments may be input in either the longform or shortform regardless of how the LONGform command is set.

The query returns the status of the LONGform command.

Command Syntax: :SYSTem:LONGform {{ON|1}|{OFF|0}}

Example:
```
OUTPUT XXX;":SYSTEM:LONGFORM ON"
```

Query Syntax: :SYSTem:LONGform?

Returned Format: [:SYSTem:LONGform] {1|0}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":SYSTEM:LONGFORM?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

MENU

command/query

The MENU command puts a menu on the display.

The MENU query returns the current menu selection.

Command Syntax: :SYSTem:MENU <menu_type>,<mach_num>

where:

<menu_type> ::= {SCONfig|FORMat|CHANnel|TRACe|TRIGger|DISPlay|WAVeform|SWAVeform|COMPare|SCHart|SLISt}

<mach_num> ::= {0 | 1 | 2 | 3}

0 ::= mixed mode

1 ::= analyzer 1

2 ::= analyzer 2

3 ::= oscilloscope

Example:

```
OUTPUT XXX;"SYSTEM:MENU FORMAT,1"
```

Query Syntax: :SYSTem:MENU?

Returned Format: [:SYSTem:MENU] <menu_type>,<mach_num>

Example:

```
10 DIM Response$[100]
20 OUTPUT XXX;":SYSTEM:MENU?"
30 ENTER XXX;Response$
40 PRINT Response$
50 END
```

MESE

command/query

The MESE command sets the Module Event Status Enable Register bits. The MESE register contains a mask value for the bits enabled in the MESR register. A one in the MESE will enable the corresponding bit in the MESR, a zero will disable the bit.

The MESE query returns the current setting.

Refer to table 6-2 for information about the Module Event Status Enable register bits, bit weights, and what each bit masks for the logic analyzer.

Command Syntax: :SYSTem:MESE <enable_mask>

where:

<enable mask> ::= integer from 0 to 255

Example:

```
OUTPUT XXX;":SYSTEM:MESE 1"
```

Query Syntax: :SYSTem:MESE?

Returned Format: [:SYSTem:MESE] <enable_mask> <NL>

Example:
```
10 OUTPUT XXX;":SYSTEM:MESE?"
20 ENTER XXX; Mes
30 PRINT Mes
40 END
```

Table 6-2. Module Event Status Enable Register

Module Event Status Enable Register (A "1" enables the MESR bit)		
Bit	Weight	Enables
7	128	Not used
6	64	Not used
5	32	Not used
4	16	Not used
3	8	Not used
2	4	Not used
1	2	RNT - Run until satisified
0	1	MC - Measurement complete

MESR

query

The MESR query returns the contents of the Module Event Status register.

Note Reading the register clears the Module Event Status Register.

Table 6-3 shows each bit in Module Event Status Register and their bit weights for the logic analyzer. When you read the MESR, the value returned is the total bit weights of all bits that are set at the time the register is read.

Query Syntax: :SYSTem:MESR?

Returned Format: [:SYSTem:MESR] <status> <NL>

where:

<status> ::= integer from 0 to 255

Example:

```
10 OUTPUT XXX;":SYSTem:MESR?"
20 ENTER XXX; Mer
30 PRINT Mer
40 END
```

Table 6-3. Module Event Status Register

Module Event Status Register		
Bit	Weight	Condition
7	128	Not used
6	64	Not used
5	32	Not used
4	16	Not used
3	8	Not used
2	4	Not used
1	2	1 = Run until satisified 0 = Run until not satisified
0	1	1 = Measurement complete 0 = Measurement not complete

PPOWer query

The PPOWer (preprocessor power) query returns the current status of the HP 1652B/53B's high-current limit circuit. If it is functioning properly, 1 is returned. If the current draw is too high, 0 is returned until the problem is corrected and the circuit automatically resets.

Query Syntax: :PPOWer?

Returned Format: [:PPOWer] {0 | 1}

Example:

```
10 DIM Response$[10]
20 OUTPUT XXX;":PPOWER?"
30 ENTER XXX; Response$
40 PRINT Response$
50 END
```

PRINt command

The PRINt command initiates a print of the screen or print all over the RS-232C bus. The PRINt parameters SCReen or ALL specify how the screen data is sent to the controller. PRINt SCReen transfers the data to the controller in a printer specific graphics format. PRINt ALL transfers the data in a raster format for the following menus:

- State and Timing Format menus
- Disk menu
- State and Timing Symbol menus
- State Listing menu
- State Trace
- State Compare

Command Syntax: :SYSTem:PRINt {SCReen|ALL}

Example:

```
OUTPUT XXX;":SYSTEM:PRINT SCREEN"
```

RMODe

command/query

The RMODe command is a run control command that specifies the run mode for logic analyzer and oscilloscope. It is at the same level in the command tree as SYSTem; therefore, it is not preceded by :SYSTem.

The query returns the current setting.

Note After specifying the run mode, use the STARt command to start the acquisition.

Command Syntax: :RMODe {SINGle|REPetitive}

Example:

```
OUTPUT XXX;":RMODE SINGLE"
```

Query Syntax: :RMODe?

Returned Format: [:RMODe] {SINGle|REPetitive}<NL>

Example:

```
10 DIM Mode$[100]
20 OUTPUT XXX;":RMODE?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

SETup command/query

The SYStem:SETup command configures the logic analyzer module as defined by the block data sent by the controller.

The SYStem:SETup query returns a block of data that contains the current configuration to the controller.

There are three data sections which are always returned and a fourth header when the oscilloscope is on and has acquired and stored waveform data. These are the strings which would be included in the section header:

- "CONFIG "
- "1650 RS232"
- "1650 DISP "
- "1650 DISP2"
- "SCOPECNF "

Additionally, the following sections may also be included, depending on what's loaded:

- "SYMBOLS A "
- "SYMBOLS B "
- "SPA DATA A"
- "SPA DATA B"
- "INVASM A "
- "INVASM B "
- "COMPARE "

Command syntax: :SYStem:SETup <block data>

where:

<block data> ::= <block length specifier> <section>...
<block length specifier> ::= #8<length>
<length> ::= the total length of all sections in byte format (must be represented with 8 digits)
<section> ::= <section header> <section data>
<section header> ::= 16 bytes in the following format:
- 10 bytes for the section name
- 1 byte reserved
- 1 byte for the module ID code (31 for the logic analyzer)
- 4 bytes for the length of the section data in bytes

<section data> ::= format depends on the type of data

Note

The total length of a section is 16 (for the section header) plus the length of the section data. So when calculating the value for <length>, don't forget to include the length of the section headers.

Example: `OUTPUT XXX USING "#,K";":SYSTEM:SETUP " <block data>`

Query Syntax: :SYStem:SETup?

Returned Format: [:SYStem:SETup] <block data><NL>

HP-IB Example:

```
10 DIM Block$[32000]              !allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":SYSTEM:HEAD OFF"
40 OUTPUT XXX;":SYSTEM:SETUP?"    ! send setup query
50 ENTER XXX USING "#,2A";Specifier$! read in #8
60 ENTER XXX USING "#,8D";Blocklength! read in block length
70 ENTER XXX USING "-K";Block$    ! read in data
80 END
```

STARt command

The STARt command is a run control command that starts the logic analyzer running in the specified run mode (see RMODe). The STARt command is on the same level in the command tree as SYSTem; therefore, it is not preceded by :SYSTem.

Note 

The STARt command is an Overlapped Command. An Overlapped Command is a command that allows execution of subsequent commands while the device operations initiated by the Overlapped Command are still in progress.

Command Syntax: :STARt

Example:

```
OUTPUT XXX;":START"
```

STOP command

The STOP command is a run control command that stops the logic analyzer. The STOP command is on the same level in the command tree as SYSTem; therefore, it is not preceded by :SYSTem.

Note

The STOP command is an Overlapped Command. An Overlapped Command is a command that allows execution of subsequent commands while the device operations initiated by the Overlapped Command are still in progress.

Command Syntax: :STOP

Example: OUTPUT XXX;":STOP"

MMEMory Subsystem 7

Introduction

MMEMory subsystem commands provide access to the disk drive. The MMEMory subsystem commands are:

- AUToload
- CATalog
- COPY
- DOWNload
- INITialize
- LOAD
- PACK
- PURGe
- REName
- STORe
- UPLoad

Note

If you are not going to store information to the configuration disk, or if the disk you are using contains information you need, it is advisable to write protect your disk. This will protect the contents of the disk from accidental damage due to incorrect commands, etc.

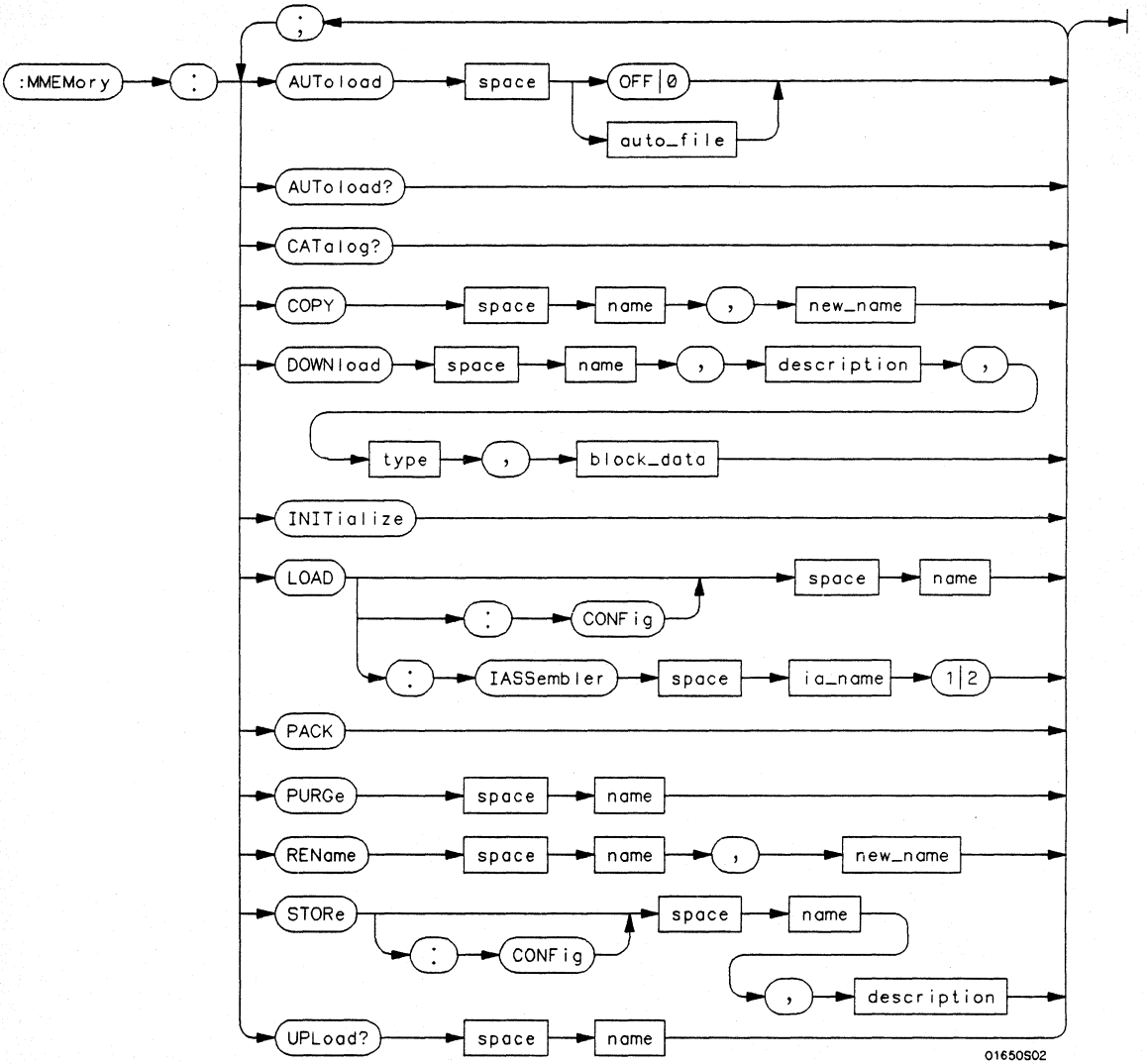


Figure 7-1. MMEMory Subsystem Commands Syntax Diagram

auto_file = *string of up to 10 alphanumeric characters representing a valid file name.*
name = *string of up to 10 alphanumeric characters representing a valid file name.*
description = *string of up to 32 alphanumeric characters.*
type = *integer, refer to table 7-1.*
block_data = *data in IEEE 488.2 # format.*
ia_name = *string of up to 10 alphanumeric characters representing a valid file name.*
new_name = *string of up to 10 alphanumeric characters representing a valid file name*

Figure 7-1. MMEMory Subsystem Commands Syntax Diagram (continued)

Note 

Refer to "Disk Operations" in chapter 5 of the *HP 1652B/53B Logic Analyzers Reference* manual for a description of a valid file name.

AUToload command/query

The AUToload command controls the autoload feature which designates a configuration file to be loaded automatically the next time the instrument is turned on. The OFF parameter (or 0) disables the autoload feature. When a string parameter is specified it represents the desired autoload file.

The AUToload query returns 0 if the autoload feature is disabled. If the autoload feature is enabled, the query returns a string parameter that specifies the current autoload file.

Command Syntax: :MMEMory:AUToload {{OFF|0}|<auto_file>}

where:

<auto_file> ::= string of up to 10 alphanumeric characters

Examples:

```
OUTPUT XXX;":MMEMORY:AUTOLOAD OFF"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE1'"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE2'"
```

Query Command: :MMEMory:AUToload?

Returned Format: [:MMEMory:AUToload] {0|<auto_file>}<NL>

Example:

```
10 DIM Auto_status$[100]
20 OUTPUT XXX;":MMEMORY:AUTOLOAD?"
30 ENTER XXX;Auto_status$
40 PRINT Auto_status$
50 END
```

CATalog query

The CATalog query returns the directory of the disk in block data format. The directory consists of a 51-character string for each file on the disk. Each file entry is formatted as follows:

"NNNNNNNNNN TTTTTTT DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD"

where N is the filename, T is the file type (a number), and D is the file description.

Query Syntax: :MMEMory:CATalog?

Returned Format: [:MMEMory:CATalog] <block size> <block data>

where:

<block size> ::= #8ddddddd (#8 followed by an eight-digit number)
<block data> ::= [<filename> <file type> <file description>]...

Example:

```
10  DIM File$[51]
20  DIM Specifier$[2]
30  OUTPUT XXX;":SYSTEM:HEAD OFF"
40  OUTPUT XXX;":MMEMORY:CATALOG?"      !send catalog query
50  ENTER XXX USING "#,2A";Specifier$   !read in #8
60  ENTER XXX USING "#,8D";Length       !read in length
70  FOR I=1 TO Length STEP 51           !read and print each file
80    ENTER XXX USING "#,51A";File$
90    PRINT File$
100   NEXT I
110 ENTER XXX USING "A";Specifier$      !read in final line feed
120 END
```

COPY command

The COPY command copies the contents of a file to a new file. The two <name> parameters are the filenames. The first parameter specifies the source file. The second specifies the destination file. An error is generated if the source file doesn't exist, if the destination file already exists, or any other disc error is detected.

Command Syntax: :MMEMory:COPY <name>,<name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name

Example: To copy the contents of "FILE1" to "FILE2":

```
OUTPUT XXX;":MMEMORY:COPY 'FILE1','FILE2'"
```

DOWNload command

The DOWNload command downloads a file to the disk. The <name> parameter specifies the filename, the <description> parameter specifies the file description, and the <block_data> contains the contents of the file to be downloaded.

Table 7-1 lists the file types for the <type> parameter.

Command Syntax: :MMEMory:DOWNload <name>,<description>,<type>,<block_data>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name
<description> ::= string of up to 32 alphanumeric characters
<type> ::= integer (see Table 7-1)
<block_data> ::= contents of file in block data format

Example:
```
OUTPUT XXX;":MMEMORY:DOWNLOAD 'SETUP__';'FILE CREATED FROM SETUP
QUERY',-16127,#800000643..."
```

Table 7-1. File Types

File	File Type
HP 1652/3 SYSTEM	-16383
1652/3 CONFIG	-16096
AUTOLOAD TYPE	-15615
INVERSE ASSEMBLER	-15614
TEXT TYPE	-15610

INITialize command

The INITialize command formats the disk.

Note

Once executed, the initialize command formats the specified disk, permanently erasing all existing information from the disk. After that, there is no way to retrieve the original information.

Command Syntax: :MMEMory:INITialize

Example:

```
OUTPUT XXX;":MMEMORY:INITIALIZE"
```

LOAD [:CONFig] command

The LOAD command loads a file from the disk into the analyzer. The [:CONfig] specifier is optional and has no effect on the command. The <name> parameter specifies the filename that will be loaded into the logic analyzer.

Note Any previous setups and data in the instrument are replaced by the contents of the configuration file.

Command Syntax: :MMEMory:LOAD[:CONfig] <name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name

Examples:

```
OUTPUT XXX;":MMEMORY:LOAD:CONFIG 'FILE__'"
OUTPUT XXX;":MMEMORY:LOAD 'FILE__'"
OUTPUT XXX;":MMEM:LOAD:CONFIG 'FILE_A'"
```

LOAD [:IASSembler] command

This variation of the LOAD command allows inverse assembler files to be loaded into analyzer 1 or analyzer 2 of the HP 1652B/1653B. The <IA_name> parameter specifies the inverse assembler filename. The parameter after the <IA_name> parameter specifies into which machine the inverse assembler is loaded.

Note 

Inverse assembler files should only be loaded into the state analyzer. If an inverse assembler file is loaded into the timing analyzer no error will be generated; however, it will not be accessible.

Command Syntax: :MMEMory:LOAD:IASSembler <IA_name>,{1|2}

where:

<IA_name> ::= string of up to 10 alphanumeric characters representing a valid file name

Examples:
```
OUTPUT XXX;":MMEMORY:LOAD:IASSEMBLER 'I68020_IP',1"
OUTPUT XXX;":MMEM:LOAD:IASS 'I68020_IP'1"
```

PACK command

The PACK command packs the files on a disk in the disk drive.

Command Syntax: :MMEMory:PACK

Example:

```
OUTPUT XXX;":MMEMORY:PACK"
```

PURGe command

The PURGe command deletes a file from the disk. The <name> parameter specifies the filename to be deleted.

Note

Once executed, the purge command permanently erases all the existing information from the specified file. After that, there is no way to retrieve the original information.

Command Syntax: :MMEMory:PURGe <name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name

Examples:

```
OUTPUT XXX;":MMEMORY:PURGE 'FILE1'"
```

RENname command

The RENname command renames a file on the disk. The <name> parameter specifies the filename to be changed and the <new_name> parameter specifies the new filename.

Note 

You cannot rename a file to an already existing filename.

Command Syntax: :MMEMory:RENname <name>,<new_name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name
<new_name> ::= string of up to 10 alphanumeric characters representing a valid file name

Examples:

```
OUTPUT XXX;":MMEMORY:RENAME 'OLDFILE','NEWFILE'"
```

STORe [:CONFig] command

The STORe command stores a configuration onto a disk. The [:CONFig] specifier is optional and has no effect on the command. The <name> parameter specifies the file to be stored to the disk. The <description> parameter specifies the file description.

Command Syntax: :MMEMory:STORe [:CONfig]<name>,<description>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name
<description> ::= string of up to 32 alphanumeric characters

Example:

```
OUTPUT XXX;":MMEM:STORE 'DEFAULTS','DEFAULT SETUPS'"
```

UPLoad

query

The UPLoad query uploads a file. The <name> parameter specifies the file to be uploaded from the disk. The contents of the file are sent out of the instrument in block data form.

Query Syntax: :MMEMory:UPLoad? <name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name

Returned Format: [:MMEMory:UPLoad] <block_data><NL>

Example:

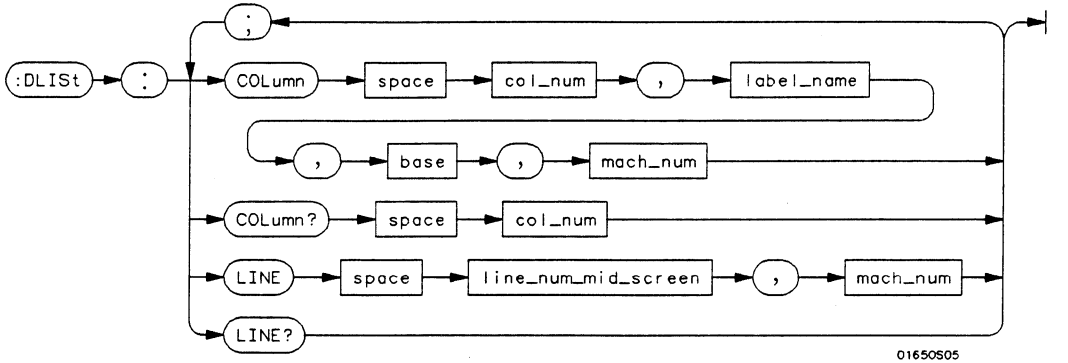
```
10 DIM Block$[32000]              !allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":SYSTEM HEAD OFF"
40 OUTPUT XXX;":MMEMORY:UPLOAD? 'FILE1'"  !send upload query
50 ENTER XXX USING "#,2A";Specifier$      !read in #8
60 ENTER XXX USING "#,8D";Length          !read in block length
70 ENTER XXX USING "-K";Block$            !read in file
80 END
```

DLISt Subsystem 8

Introduction

The DLISt (dual list) subsystem contains the commands in the dual state listing menu. These commands are:

- COLumn
- LINE



col_num = *integer from 1 to 8*
label_name = *a string of up to 6 alphanumeric characters*
base = {*BINary*|*HEXacecimal*|*OCTal*|*DECimal*|*ASCii*|*SYMBol*}
mach_num = {*1*|*2*}
line_num_mid_screen = *integer from -1023 to +1023*

Figure 8-1. DLISt Subsystem Syntax Diagram

DLISt selector

The DLISt selector (dual list) is used as part of a compound header to access those settings normally found in the Dual State Listing menu. The dual list displays data when two state analyzers are run simultaneously.

Command Syntax: :DLISt

Example:

```
OUTPUT XXX;":DLIST:LINE 0,1"
```

COLumn

command/query

The COLumn command allows you to configure the state analyzer list display by assigning a label name and base to one of eight vertical columns in the menu. The machine number parameter is required since the same label name can occur in both state machines at once. A column number of 1 refers to the left-most column. When a label is assigned to a column it replaces the original label in that column. The label originally in the specified column is placed in the column the specified label is moved from.

When "TAGS" is the label name, the TAGS column is assumed and the next parameter must specify RELative or ABSolute. The machine number should be 1.

The COLumn query returns the column number, label name, and base for the specified column.

Command Syntax: :DLISt:COLumn <col_num>,{"TAGS",{RELative|ABSolute}|<label_name>,<base>},<mach_num>

where:

<col_num> ::= {1|2|3|4|5|6|7|8}

<label_name> ::= a string of up to 6 alphanumeric characters

<base> ::= {BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol}

<mach_num> ::= {1|2}

Example:

```
OUTPUT XXX;":DLIST:COLUMN 4,'DATA',HEXADECIMAL,1"
```

COLumn

Query Syntax: :DLISt:COLumn? <col_num>

Returned Format: [:DLISt:COLumn] <col_num>,<label_name>,<base>,<mach_num><NL>

Example:

```
10 DIM C1$[100]
20 OUTPUT XXX;":DLIST:COLUMN? 4"
30 ENTER XXX;C1$
40 PRINT C1$
50 END
```

LINE

command/query

The LINE command allows you to scroll the state analyzer listing vertically. The command specifies the state line number relative to the trigger that the specified analyzer will highlight at center screen.

The LINE query returns the line number for the state currently in the box at center screen and the machine number to which it belongs.

Command Syntax: :DLISt:LINE <line_num_mid_screen>,<mach_num>

where:

<line_num_mid_screen> ::= integer from -1023 to +1023
<mach_num> ::= {1|2}

Example:

```
OUTPUT XXX;":DLIST:LINE 511,1"
```

Query Syntax: :DLISt:LINE?

Returned Format: [DLISt:LINE] <line_num_mid_screen>,<mach_num><NL>

Example:

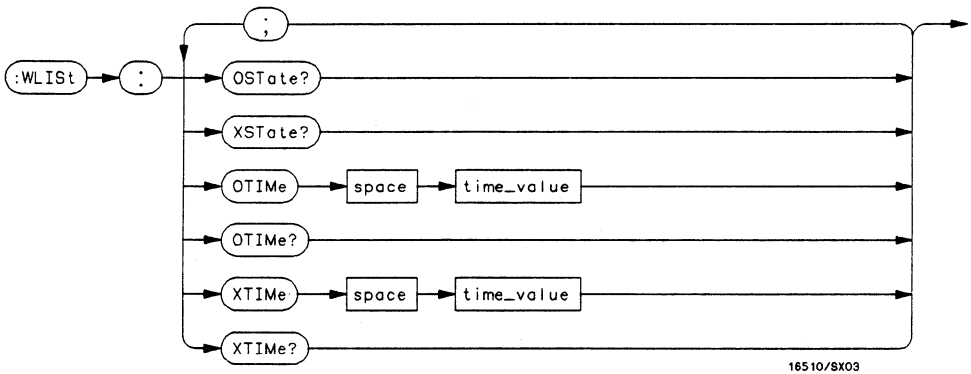
```
10 DIM Ln$[100]
20 OUTPUT XXX;":DLIST:LINE?"
30 ENTER XXX;Ln$
40 PRINT Ln$
50 END
```

WLISt Subsystem 9

Introduction

Two commands in the WLISt subsystem control the X and O marker placement on the waveforms portion of the Timing/State mixed mode display. These commands are XTIMe and OTIMe. The XSTate and OSTate queries return what states the X and O markers are on. Since the markers can only be placed on the timing waveforms, the queries return what state (state acquisition memory location) the marked pattern is stored in.

Note In order to have mixed mode, one machine must be a timing analyzer and the other must be a state analyzer with time tagging on (use MACHine<N>:STRace:TAG TIME).



time_value = *real number*

Figure 9-1. WLISt Subsystem Syntax Diagram

WLISt selector

The WLISt (Waveforms/listing) selector is used as a part of a compound header to access the settings normally found in the Mixed Mode menu. Since the WLISt command is a root level command, it will always appear as the first element of a compound header.

Note The WLISt Subsystem is only available when one state analyzer (with time tagging on) and one timing analyzer are specified.

Command Syntax: :WLISt

Example:

```
OUTPUT XXX;":WLIST:XTIME 40.0E-6"
```

OSTate query

The OSTate query returns the state where the O Marker is positioned. If data is not valid, the query returns 32767.

Query Syntax: :WLISt:OSTate?

Returned Format: [:WLISt:OSTate] <state_num> <NL>

where:

<state_num> ::= integer

Example:

```
10 DIM So$[100]
20 OUTPUT XXX;":WLIST:OSTATE?"
30 ENTER XXX;So$
40 PRINT So$
50 END
```

XSTate query

The XSTate query returns the state where the X Marker is positioned. If data is not valid, the query returns 32767.

Query Syntax: :WLISt:XSTate?

Example:

```
OUTPUT XXX,":WLIST:XSTATE?
```

Returned Format: [:WLISt:XSTate] <state_num> <NL>

where:

<state_num> ::= integer

Example:

```
10 DIM Sx$[100]
20 OUTPUT XXX;":WLIST:XSTATE?"
30 ENTER XXX;Sx$
40 PRINT Sx$
50 END
```

OTIMe command/query

The OTIMe command positions the O Marker on the timing waveforms in the mixed mode display. If the data is not valid, the command performs no action.

The OTIMe query returns the O Marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :WLISt:OTIMe <time_value>

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX,":WLIST:OTIME 40.0e-6"
```

Query Syntax: :WLISt:OTIMe?

Returned Format: [:WLISt:OTIMe] <time_value><NL>

Example:
```
10 DIM To$[100]
20 OUTPUT XXX;":WLIST:OTIME?"
30 ENTER XXX;To$
40 PRINT To$
50 END
```

XTIMe command/query

The XTIMe command positions the X Marker on the timing waveforms in the mixed mode display. If the data is not valid, the command performs no action.

The XTIMe query returns the X Marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :WLISt:XTIMe <time_value>

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX,":WLIST:XTIME 40.0E-6"
```

Query Syntax: :WLISt:XTIMe?

Returned Format: [:WLISt:XTIMe] <time_value><NL>

Example:
```
10 DIM Tx$[100]
20 OUTPUT XXX;":WLIST:XTIME?"
30 ENTER XXX;Tx$
40 PRINT Tx$
50 END
```

MACHine Subsystem 10

Introduction

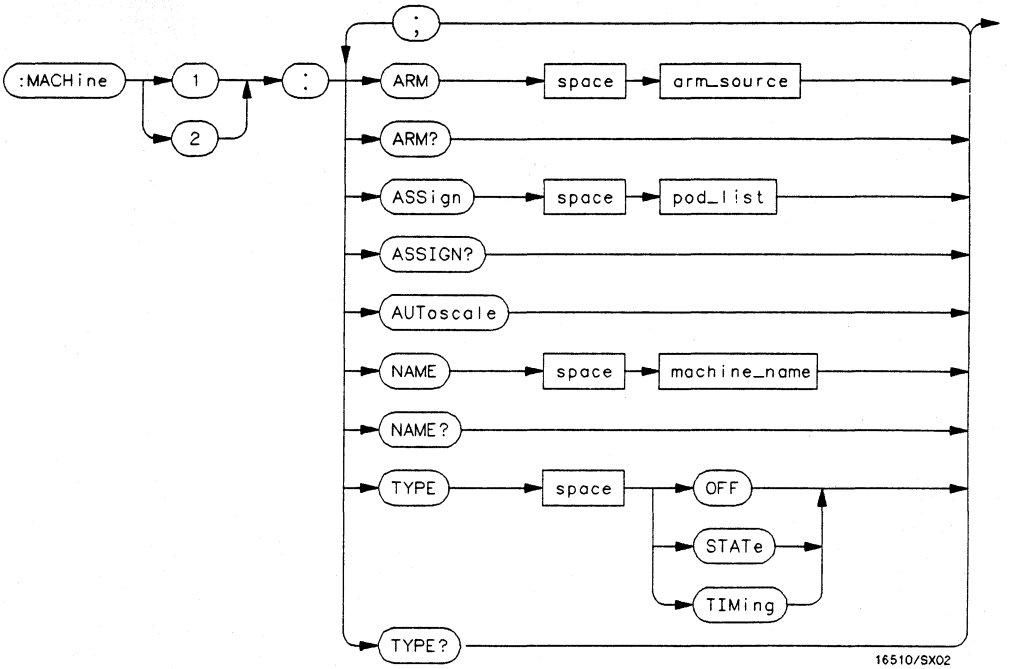
The MACHine subsystem contains the commands available for the State/Timing Configuration menu. These commands are:

- ARM
- ASSign
- AUToscale (Timing Analyzer only)
- NAME
- TYPE

There are actually two MACHine subsystems: MACHine1 and MACHine2. Unless noted, they are identical. In the syntax definitions you will see MACHine{1|2} anytime the subject is applicable to both subsystems.

Additionally, the following subsystems are a part of the MACHine subsystem. Each is explained in a separate chapter.

- SFORmat subsystem (chapter 11)
- STRace subsystem (chapter 12)
- SLISt subsystem (chapter 13)
- SWAVeform subsystem (chapter 14)
- SCHart subsystem (chapter 15)
- COMPare subsystem (chapter 16)
- TFORmat subsystem (chapter 17)
- TTRace subsystem (chapter 18)
- TWAVeform subsystem (chapter 19)
- SYMBol subsystem (chapter 20)



arm_source = {*RUN* | *MACHine* {*1* | *2*}}
pod_list = {*NONE* | <*pod_num*>[, <*pod_num*>]...}
pod_num = {*1* | *2* | *3* | *4* | *5*}
machine_name = *string of up to 10 alphanumeric characters*

Figure 10-1. Machine Subsystem Syntax Diagram

MACHine selector

The MACHine < N > selector specifies which of the two analyzers (machines) available in the HP 1652B/53B the commands or queries following will refer to. Since the MACHine < N > command is a root level command, it will normally appear as the first element of a compound header.

Command Syntax: :MACHine < N >

where:

< N > ::= {1|2} (the number of the machine)

Example:

```
OUTPUT XXX; ":MACHINE1:NAME 'DRAMTEST'"
```

ARM command/query

The ARM command specifies the arming source of the specified analyzer (machine).

The ARM query returns the source that the current analyzer (machine) will be armed by.

Command Syntax: :MACHine{1|2}:ARM <arm_source>

where:

<arm_source> ::= {RUN|MACHine{1|2}|BNC|SCOPe}

Example:

```
OUTPUT XXX;":MACHINE1:ARM MACHINE2"
```

Query Syntax: :MACHine {1|2}:ARM?

Returned Format: [:MACHine {1|2}:ARM] <arm_source> <NL>

Example:

```
10 DIM String$ [100]
20 OUTPUT XXX; ":MACHINE1:ARM?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

ASSign command/query

The ASSign command assigns pods to a particular analyzer (machine).

The ASSign query returns which pods are assigned to the current analyzer (machine).

Command Syntax: :MACHine{1|2}:ASSign <pod_list>

where:

<pod_list> ::= {NONE|<pod #>[, <pod #>]...}
<pod #> ::= {1|2|3|4|5}

Example:
```
OUTPUT XXX;":MACHINE1:ASSIGN 5, 2, 1"
```

Query Syntax: :MACHine {1|2}:ASSign?

Returned Format: [:MACHINE {1|2}:ASSign] <pod_list><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:ASSIGN?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

AUToscale command

The AUToscale command causes the current analyzer (machine) to autoscale if the current machine is a timing analyzer. If the current machine is not a timing analyzer, the AUToscale command is ignored.

AUToscale is an Overlapped Command. Overlapped Commands allow execution of subsequent commands while the logic analyzer operations initiated by the Overlapped Command are still in progress. Command overlapping can be avoided by using the *OPC and *WAI commands in conjunction with AUToscale (see chapter 5, "Common Commands.")

Note When the AUToscale command is issued, existing timing analyzer configurations are erased and the other analyzer is turned off.

Command Syntax: :MACHine{1|2}:AUToscale

Example:

```
OUTPUT XXX;":MACHINE1:AUTOSCALE"
```

NAME

command/query

The NAME command allows you to assign a name of up to 10 characters to a particular analyzer (machine) for easier identification.

The NAME query returns the current analyzer name as an ASCII string.

Command Syntax: :MACHine{1|2}:NAME <machine_name>

where:

<machine_name> ::= string of up to 10 alphanumeric characters

Example:

```
OUTPUT XXX;":MACHINE1:NAME 'DRAMTEST'"
```

Query Syntax: :MACHine{1|2}:NAME?

Returned Format: [MACHine{1|2}:NAME] <machine name><NL>

Example:

```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:NAME?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TYPE command/query

The TYPE command specifies what type a specified analyzer (machine) will be. The analyzer types are state or timing. The TYPE command also allows you to turn off a particular machine.

Note 

Only one of the two analyzers can be specified as a timing analyzer at one time.

The TYPE query returns the current analyzer type for the specified analyzer.

Command Syntax: :MACHine{1|2}:TYPE <analyzer type>

where:

<analyzer type> ::= {OFF|STATe|TIMing}

Example:
```
OUTPUT XXX;":MACHINE1:TYPE STATE"
```

Query Syntax: :MACHine{1|2}:TYPE?

Returned Format: [:MACHine{1|2}:TYPE] <analyzer type><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:TYPE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SFORmat Subsystem 11

Introduction

The SFORmat subsystem contains the commands available for the State Format menu in the HP 1652B/53B logic analyzer. These commands are:

- CLOCk
- CPERiod
- LABel
- MASTer
- REMove
- SLAVe
- THReshold

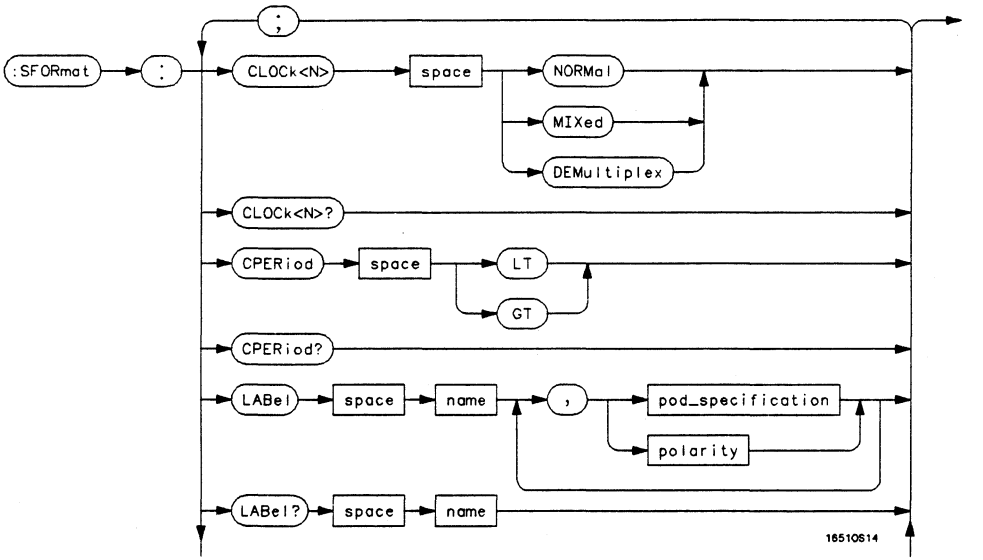
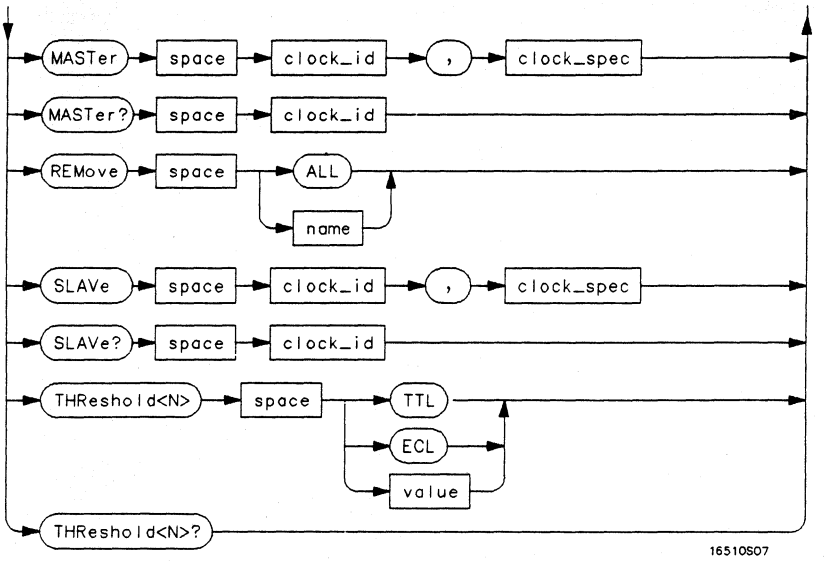


Figure 11-1. SFORmat Subsystem Syntax Diagram



<N> = *{1 | 2 | 3 | 4 | 5}*
GT = *Greater Than 60 ns*
LT = *Less Than 60 ns*
name = *string of up to 6 alphanumeric characters*
polarity = *{POSitive | NEGative}*
pod_specification = *format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)*
clock_id = *{J | K | L | M | N}*
clock_spec = *{OFF | RISing | FALLing | BOTH | LOW | HIGH}*
value = *voltage (real number) -9.9 to +9.9*

Figure 11-1. SFORmat Subsystem Syntax Diagram (continued)

SFORmat selector

The SFORmat (State Format) selector is used as a part of a compound header to access the settings in the State Format menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SFORmat

Example:

```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER J, RISING"
```

CLOCk command/query

The CLOCk command selects the clocking mode for a given pod when the pod is assigned to the state analyzer. When the NORMal option is specified, the pod will sample all 16 channels on the master clock. When the MIXed option is specified, the upper 8 bits will be sampled by the master clock and the lower 8 bits will be sampled by the slave clock. When the DEMultiplex option is specified, the lower 8 bits will be sampled on the slave clock and then sampled again on the master clock. The master clock always follows the slave clock when both are used.

The CLOCk query returns the current clocking mode for a given pod.

Command Syntax: :MACHine{1|2}:SFORmat:CLOCk<N> <clock_mode>

where:

<N> ::= Pod {1|2|3|4|5}
<clock_mode> ::= {NORMal | MIXed | DEMultiplex}

Example:
```
OUTPUT XXX;":MACHINE1:SFORMAT:CLOCK2 NORMAL"
```

Query Syntax: :MACHine{1|2}:SFORmat:CLOCk<N>?

Returned Format: [:MACHine{1|2}:SFORmat:CLOCK<N>] <clock_mode><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX; ":MACHINE1:SFORMAT:CLOCK2?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

CPERiod command/query

The CPERiod command allows you to set the state analyzer for input clock periods of greater than or less than 60 ns. Either LT or GT can be specified. LT signifies a state input clock period of less than 60 ns, and GT signifies a period of greater than 60 ns.

Because count tagging requires a minimum clock period of 60 ns, the CPERiod and TAG commands are interrelated (the TAG command is in the STRace subsystem). When the clock period is set to Less Than, count tagging is turned off. When count tagging is set to either state or time, the clock period is automatically set to Greater Than.

The CPERiod query returns the current setting of clock period.

Command Syntax: :MACHine{1|2}:SFORmat:CPERiod {LT|GT}

where:

GT ::= greater than 60 ns
LT ::= less than 60 ns

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:CPERIOD GT"
```

Query Syntax: :MACHine{1|2}:SFORmat:CPERiod?

Returned Format: [:MACHine{1|2}:SFORmat:CPERiod] {GT|LT}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:CPERIOD?
30 ENTER XXX; String$
40 PRINT String$
50 END
```

LABel command/query

The LABel command allows you to specify polarity and assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label will be created.

The order of the pod-specification parameters is significant. The first one listed will match the highest-numbered pod assigned to the machine you're using. Each pod specification after that is assigned to the next-highest-numbered pod. This way they match the left-to-right descending order of the pods you see on the Format display. Not including enough pod specifications results in the lowest-numbered pod(s) being assigned a value of zero (all channels excluded). If you include more pod specifications than there are pods for that machine, the extra ones will be ignored. However, an error is reported anytime more than five pod specifications are listed.

The polarity can be specified at any point after the label name.

Since pods contain 16 channels, the format value for a pod must be between 0 and 65535 (2^{16}-1). When giving the pod assignment in binary (base 2), each bit will correspond to a single channel. A "1" in a bit position means the associated channel in that pod is assigned to that pod and bit. A "0" in a bit position means the associated channel in that pod is excluded from the label. For example, assigning #B1111001100 is equivalent to entering "......****..**.." through the front-panel user interface.

A label can not have a total of more than 32 channels assigned to it.

The LABel query returns the current specification for the selected (by name) label. If the label does not exist, nothing is returned. The polarity is always returned as the first parameter. Numbers are always returned in decimal format.

Command Syntax: :MACHine{1|2}:SFORmat:LABel <name>[,{<polarity> | <assignment>}]...

where:

<name> ::= string of up to 6 alphanumeric characters
<polarity> ::= {POSitive | NEGative}
<assignment> ::= format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

Examples:
```
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'STAT', POSITIVE, 65535,127,40312"
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'SIG 1', 64, 12, 0, 20, NEGATIVE"
OUTPUT XXX;":MACHINE1:SFORMAT:LABEL 'ADDR', NEG, #B0011110010101010"
```

Query Syntax: :MACHine{1|2}:SFORmat:LABel?<name>

Returned Format: [:MACHine{1|2}:SFORmat:LABel] <name>,<polarity>[,<assignment>]...<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:LABEL? 'DATA'"
30 ENTER XXX String$
40 PRINT String$
50 END
```

MASTer command/query

The MASTer clock command allows you to specify a master clock for a given machine. The master clock is used in all clocking modes (Normal, Mixed, and Demultiplexed). Each command deals with only one clock (J,K,L,M,N); therefore, a complete clock specification requires five commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed. Level specifications (LOW or HIGH) are ANDed.

Note At least one clock edge must be specified.

The MASTer query returns the clock specification for the specified clock.

Command Syntax: :MACHine{1|2}:SFORmat:MASTer <clock_id>,<clock_spec>

where:

<clock_id> ::= {J|K|L|M|N}
<clock_spec> ::= {OFF|RISing|FALLing|BOTH|LOW|HIGH}

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER J, RISING"
```

Query Syntax: :MACHine{1|2}:SFORmat:MASTer? <clock_id>

Returned Format: [:MACHine{1|2}:SFORmat:MASTer] <clock_id>,<clock_spec><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:MASTER?<clock_id>"
30 ENTER XXX String$
40 PRINT String$
50 END
```

REMove command

The REMove command allows you to delete all labels or any one label for a given machine.

Command Syntax: :MACHine{1|2}:SFORmat:REMove {<name>|ALL}

where:

<name> ::= string of up to 6 alphanumeric characters

Examples:

```
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE 'A'"
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE ALL"
```

SLAVe command/query

The SLAVe clock command allows you to specify a slave clock for a given machine. The slave clock is only used in the Mixed and Demultiplexed clocking modes. Each command deals with only one clock (J,K,L,M,N); therefore, a complete clock specification requires five commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed. Level specifications (LOW or HIGH) are ANDed.

Note The slave clock must have at least one edge specified.

The SLAVe query returns the clock specification for the specified clock.

Command Syntax: :MACHine{1|2}:SFORmat:SLAVe <clock_id>,<clock_spec>

where:

<clock_id> ::= {J|K|L|M|N}
<clock_spec> ::= {OFF|RISing|FALLing|BOTH|LOW|HIGH}

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE J, RISING"
```

Query Syntax: :MACHine{1|2}:SFORmat:SLAVe?<clock_id>

Returned Format: [:MACHine{1|2}:SFORmat:SLAVe] <clock_id>,<clock_spec><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE? <clock_id>"
30 ENTER XXX String$
40 PRINT String$
50 END
```

THReshold command/query

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL, or a specific voltage from -9.9V to +9.9V in 0.1 volt increments.

Note 

On the HP 1652B, the pod thresholds of pods 1, 2 and 3 can be set independently. The pod thresholds of pods 4 and 5 are slaved together; therefore, when you set the threshold on either pod 4 or 5, both thresholds will be changed to the specified value. On the HP 1653B, pods 1 and 2 can be set independently.

The THReshold query returns the current threshold for a given pod.

Command Syntax: :MACHine{1|2}:SFORmat:THReshold<N> {TTL|ECL|<value>}

where:

<N> ::= pod number {1|2|3|4|5}
<value> ::= voltage (real number) -9.9 to +9.9
TTL ::= default value of +1.6V
ECL ::= default value of -1.3V

Example:
```
OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD1 4.0"
```

Query Syntax: :MACHine{1|2}:SFORmat:THReshold<N>?

Returned Format: [:MACHine{1|2}:SFORmat:THReshold<N>] <value><NL>

Example:
```
10 DIM Value$ [100]
20 OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD4?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

STRace Subsystem 12

Introduction

The STRace subsystem contains the commands available for the State Trace menu in the HP 1652B/53B logic analyzer. The STRace subsystem commands are:

- BRANch
- FIND
- PREStore
- RANGe
- RESTart
- SEQuence
- STORe
- TAG
- TERM

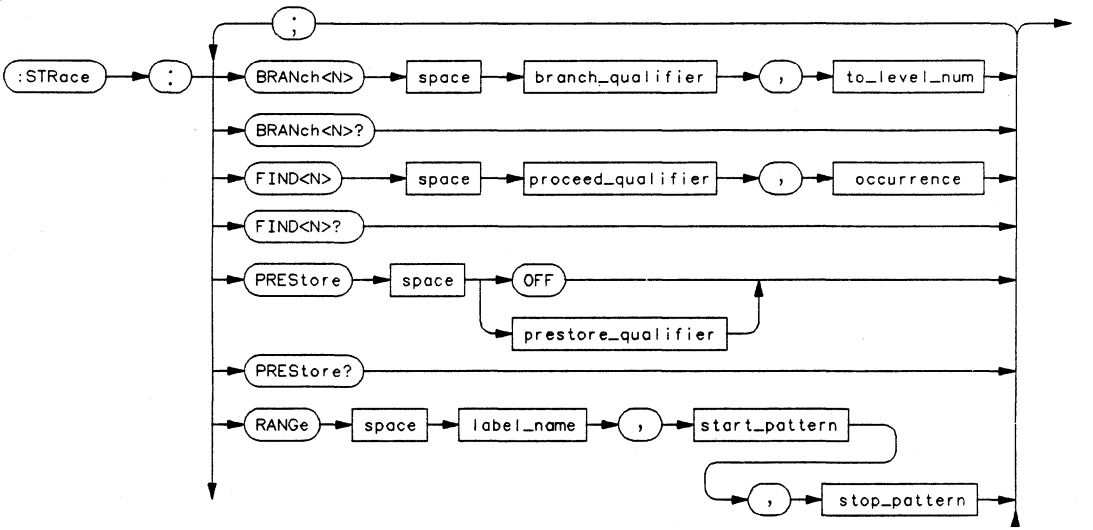


Figure 12-1. STRace Subsystem Syntax Diagram

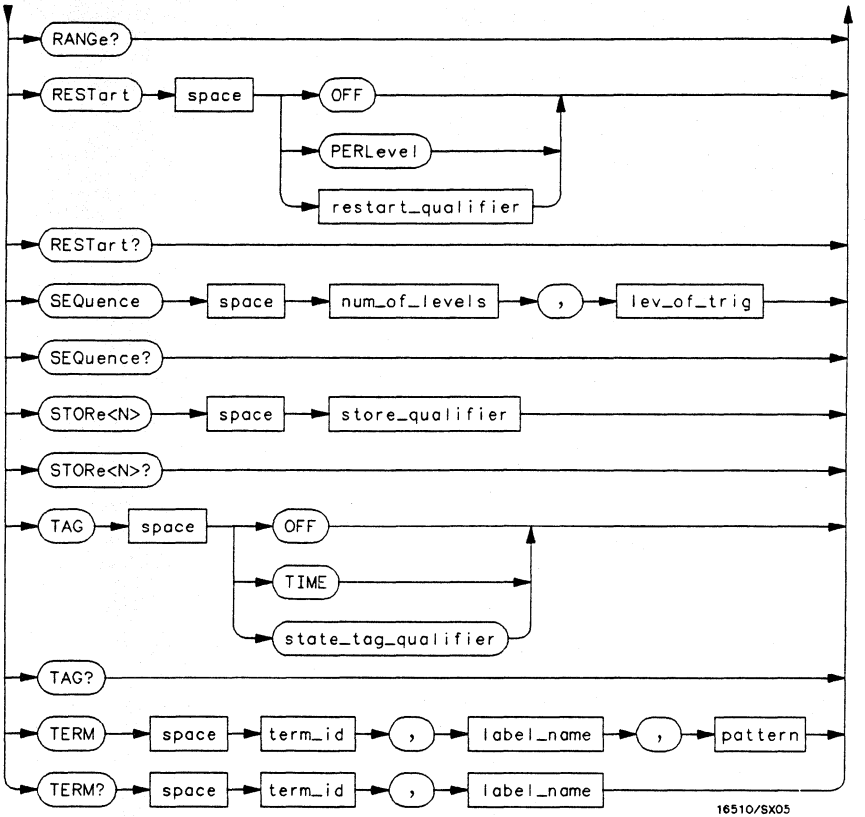


Figure 12-1. STRace Subsystem Syntax Diagram (continued)

branch_qualifier = *<qualifier>*
to_lev_num = *integer from 1 to trigger level when <N> is less than or equal to the trigger level, or from (trigger level + 1) to <num_of_levels> when <N> is greater than the trigger level*
proceed_qualifier = *<qualifier>*
occurrence = *number from 1 to 65535*
prestore_qual = *<qualifier>*
label_name = *string of up to 6 alphanumeric characters*
start_pattern = "{#B{0|1}... |
#Q{0|1|2|3|4|5|6|7}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}... |
{0|1|2|3|4|5|6|7|8|9}... }"
stop_pattern = "{#B{0|1}... |
#Q{0|1|2|3|4|5|6|7}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}... |
{0|1|2|3|4|5|6|7|8|9}... }"
restart_qualifier = *<qualifier>*
num_of_levels = *integer from 2 to 8 when ARM is RUN or from 2 to 7 otherwise*
lev_of_trig = *integer from 1 to (number of existing sequence levels - 1)*
store_qualifier = *<qualifier>*
state_tag_qualifier = *<qualifier>*
term_id = {A|B|C|D|E|F|G|H}
pattern = "{#B{0|1|X}... |
#Q{0|1|2|3|4|5|6|7|X}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}... |
{0|1|2|3|4|5|6|7|8|9}... }"
qualifier = { *ANYState* | *NOSTate* | *<any_term>* | *(expression1[{AND|OR} <expression2>])* |
(expression2[{AND|OR} <expression1>]) }
any_term = {*<or_term1>* | *<and_term1>* | *<or_term2>* | *and_term2*}
expression1 = {*<or_term1>[OR <or_term1>]...* | *<and_term1>[AND <and_term1>]...*}
expression2 = {*<or_term2>[OR <or_term2>]...* | *<and_term2>[AND <and_term2>]...*}
or_term1 = {A|B|C|D|*INRange*|*OUTRange*}
and_term1 = {*NOTA*|*NOTB*|*NOTC*|*NOTD*|*INRange*|*OUTRange*}
or_term2 = {E|F|G|H}
and_term2 = {*NOTE*|*NOTF*|*NOTG*|*NOTH*}

Figure 12-1. STRace Subsystem Syntax Diagram (continued)

STRace selector

The STRace (State Trace) selector is used as a part of a compound header to access the settings found in the State Trace menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:STRace

Example:

```
OUTPUT XXX;":MACHINE1:STRACE:TAG TIME"
```

BRANch command/query

The BRANch command defines the branch qualifier for a given sequence level. When this branch qualifier is matched, it will cause the sequencer to jump to the specified sequence level.

Note "RESTART PERLEVEL" must have been invoked for this command to have an effect (see RESTart command).

The terms used by the branch qualifier (A through H) are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command.

Within the limitations shown by the syntax definitions, complex expressions may be formed using the AND and OR operators. Expressions are limited to what you could manually enter through the front panel. Regarding parentheses, the syntax definitions on the next page show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. For example, the following two statements are both correct and have the same meaning. Notice that the conventional rules for precedence are not followed.

```
OUTPUT XXX;":MACHINE1:STRACE:BRANCH1 (C OR D AND F OR G), 1"
OUTPUT XXX;":MACHINE1:STRACE:BRANCH1 ((C OR D) AND (F OR G)), 1"
```

Figure 12-2 shows a complex expression as seen on the Format display.

Note Branching across the trigger level is not allowed. Therefore, the values for <N> and <to_level_num> must both be either on or before the trigger level, or they must both be after the trigger level. The trigger level is determined through the SEQuence command.

The BRANch query returns the current branch qualifier specification for a given sequence level.

BRANch

Command Syntax: :MACHine{1|2}:STRace:BRANch<N> <branch_qualifier>,<to_level_number>

where:

<N> ::= an integer from 1 to <number_of_levels>

<to_level_number> ::= integer from 1 to trigger level, when <N> is less than or equal to the trigger level or from (trigger level + 1) to <number_of_levels>, when <N> is greater than the trigger level

<number_of_levels> ::= integer from 2 to the number of existing sequence levels (maximum 8)

<branch_qualifier> ::= { ANYState | NOSTate | <any_term> |
(<expression1>[{AND|OR} <expression2>]) |
(<expression2>[{AND|OR} <expression1>]) }

<any_term> ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}

<expression1> ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}

<expression2> ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}

<or_term1> ::= {A|B|C|D|INRange|OUTRange}

<and_term1> ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}

<or_term2> ::= {E|F|G|H}

<and_term2> ::= {NOTE|NOTF|NOTG|NOTH}

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:BRANCH1 ANYSTATE, 3"
OUTPUT XXX;":MACHINE2:STRACE:BRANCH2 A, 7"
OUTPUT XXX;":MACHINE1:STRACE:BRANCH3 ((A OR B) OR NOTG), 1"
```

Query Syntax :MACHine{1|2}:STRace:BRANch<N>?

Returned Format: [:MACHine{1|2}:STRace:BRANch<N>]<branch_qualifier>,<to_level_num><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:BRANCH3?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

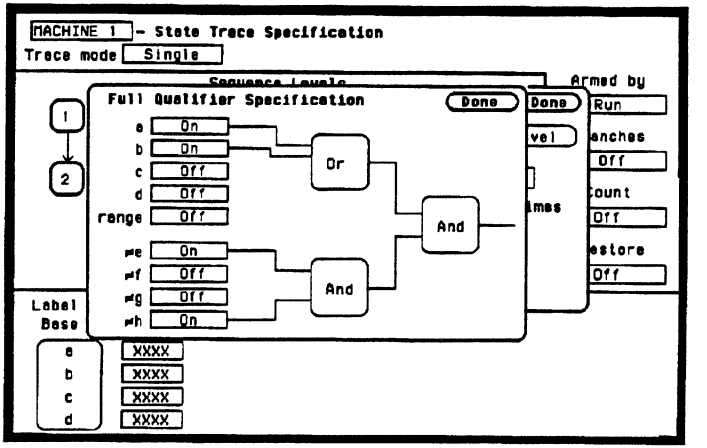


Figure 12-2. Complex qualifier

Figure 12-2 is a front panel representation of the complex qualifier **(a Or b) And (≠e And ≠h)**. The following example would be used to specify this complex qualifier.

```
OUTPUT XXX;":MACHINE1:STRACE:BRANCH1 ((A OR B) AND (NOTE AND NOTH)), 2"
```

Note

Terms **A** through **D** and **RANGE** must be grouped together and terms **E** through **H** must be grouped together. In the first level, terms from one group may not be mixed with terms from the other. For example, the expression ((A OR INRANGE) AND (C OR H)) is not allowed because the term C cannot be specified in the **E** through **H** group.

Keep in mind that, at the first level, the operator you use determines which terms are available. When AND is chosen, only the **NOT** terms may be used. Either AND or OR may be used at the second level to join the two groups together. It is acceptable for a group to consist of a single term. Thus, an expression like (B AND G) is legal, since the two operands are both simple terms from separate groups.

FIND command/query

The FIND command defines the proceed qualifier for a given sequence level. The qualifier tells the state analyzer when to proceed to the next sequence level. When this proceed qualifier is matched the specified number of times, the sequencer will proceed to the next sequence level. The state that causes the sequencer to switch levels is automatically stored in memory whether it matches the associated store qualifier or not. In the sequence level where the trigger is specified, the FIND command specifies the trigger qualifier (see SEQuence command).

The terms A through H are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command. Expressions are limited to what you could manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. See figure 6-2 for a detailed example.

The FIND query returns the current proceed qualifier specification for a given sequence level.

Command Syntax: :MACHine{1|2}:STRace:FIND<N> <proceed_qualifier>,<occurrence>

where:

<N> ::= integer from 1 to the number of existing sequence levels (maximum 8)

<occurrence> ::= integer from 1 to 65535

<proceed_qualifier> ::= { ANYState | NOSTate | <any_term> |
(<expression1>[{AND|OR} <expression2>]) |
(<expression2>[{AND|OR} <expression1>]) }

<any_term> ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}

<expression1> ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}

<expression2> ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}

<or_term1> ::= {A|B|C|D|INRange|OUTRange}

<and_term1> ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}

<or_term2> ::= {E|F|G|H}

<and_term2> ::= {NOTE|NOTF|NOTG|NOTH}

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:FIND1 ANYSTATE, 1"
OUTPUT XXX;":MACHINE1:STRACE:FIND2 A, 512"
OUTPUT XXX;":MACHINE1:STRACE:FIND3 ((NOTA AND NOTB) OR G), 1"
```

Query Syntax: :MACHine{1|2}:STRace:FIND4?

Returned Format: [:MACHine{1|2}:STRace:FIND<N>] <proceed_qualifier>,<occurrence><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:FIND<N>?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

PREStore command/query

The PREStore command turns the prestore feature on and off. It also defines the qualifier required to prestore only selected states. The terms A through H are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command.

Expressions are limited to what you could manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed.

A detailed example is provided in figure 12-2.

The PREStore query returns the current prestore specification.

Command Syntax: :MACHine{1|2}:STRace:PREStore {OFF | <prestore_qualifier>}

where:

```
<prestore_qualifier> ::= { ANYState | NOSTate | <any_term> |
                           (<expression1>[{AND|OR} <expression2>]) |
                           (<expression2>[{AND|OR} <expression1>]) }
        <any_term> ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
     <expression1> ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
     <expression2> ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
        <or_term1> ::= {A|B|C|D|INRange|OUTRange}
       <and_term1> ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
        <or_term2> ::= {E|F|G|H}
       <and_term2> ::= {NOTE|NOTF|NOTG|NOTH}
```

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE OFF"
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE ANYSTATE"
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE (E)"
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE (A OR B OR D OR F OR H)"
```

Query Syntax: :MACHine{1|2}:STRace:PREStore?

Returned Format: [:MACHine{1|2}:STRace:PREStore] {OFF|<prestore_qualifier>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:PRESTORE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

RANGe command/query

The RANGe command allows you to specify a range recognizer term in the specified machine. Since a range can only be defined across one label and, since a label must contain 32 or less bits, the value of the start pattern or stop pattern will be between (2^{32})-1 and 0.

Note

Since a label can only be defined across a maximum of two pods, a range term is only available across a single label; therefore, the end points of the range cannot be split between labels.

When these values are expressed in binary, they represent the bit values for the label at one of the range recognizers' end points. Don't cares are not allowed in the end point pattern specifications. Since only one range recognizer exists, it is always used by the first state machine defined.

The RANGe query returns the range recognizer end point specifications for the range.

Note

When two state analyzers are on, the RANGe term is not available in the second state analyzer assigned and there are only 4 pattern recognizers per analyzer.

Command Syntax: :MACHine{1|2}:STRace:RANGE <label_name>,<start_pattern>,<stop_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<start_pattern> ::= "{#B{0|1}...|
#Q{0|1|2|3|4|5|6|7}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...|
{0|1|2|3|4|5|6|7|8|9}...}"

<stop_pattern> ::= "{#B{0|1}...|
#Q{0|1|2|3|4|5|6|7}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...|
{0|1|2|3|4|5|6|7|8|9}...}"

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:RANGE 'DATA', '127', '255' "
OUTPUT XXX;":MACHINE1:STRACE:RANGE 'ABC', '#B00001111', '#HCF' "
```

Query Syntax: :MACHine{1|2}:STRace:RANGe?

Returned Format: [:MACHine{1|2}:STRAce:RANGe]
<label_name>,<start_pattern>,<stop_pattern><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:RANGE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

RESTart command/query

The RESTart command selects the type of restart to be enabled during the trace sequence. It also defines the global restart qualifier that restarts the sequence in global restart mode. The qualifier may be a single term or a complex expression. The terms A through H are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command.

Expressions are limited to what you could manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed.

A detailed example is provided in figure 12-2.

The RESTart query returns the current restart specification.

Command Syntax: :MACHine{1|2}:STRace:RESTart {OFF | PERLevel | <restart_qualifier>}

where:

<restart_qualifier> ::= { ANYState | NOSTate | <any_term> |
(<expression1>[{AND|OR} <expression2>]) |
(<expression2>[{AND|OR} <expression1>]) }

<any_term> ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}

<expression1> ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}

<expression2> ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}

<or_term1> ::= {A|B|C|D|INRange|OUTRange}

<and_term1> ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}

<or_term2> ::= {E|F|G|H}

<and_term2> ::= {NOTE|NOTF|NOTG|NOTH}

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:RESTART OFF"
OUTPUT XXX;":MACHINE1:STRACE:RESTART PERLEVEL"
OUTPUT XXX;":MACHINE1:STRACE:RESTART (NOTA AND NOTB AND INRANGE)"
OUTPUT XXX;":MACHINE1:STRACE:RESTART (B OR (NOTE AND NOTF))"
```

Query Syntax: :MACHine{1|2}:STRace:RESTart?

Returned Format: [:MACHine{1|2}:STRace:RESTart] {OFF | PERLevel | <restart_qualifier>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:RESTART?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SEQuence command/query

The SEQuence command redefines the state analyzer trace sequence. First, it deletes the current trace sequence. Then it inserts the number of levels specified, with default settings, and assigns the trigger to be at a specified sequence level. The number of levels can be between 2 and 8 when the analyzer is armed by the RUN key. When armed by the BNC or the other machine, a level is used by the arm in; therefore, only seven levels are available in the sequence.

The SEQuence query returns the current sequence specification.

Command Syntax: :MACHine{1|2}:STRace:SEQuence <number_of_levels>,<level_of_trigger>

where:

<number_of_levels> ::= integer from 2 to 8 when ARM is RUN or from 2 to 7 otherwise
<level_of_trigger> ::= integer from 1 to (number of existing sequence levels - 1)

Example:
```
OUTPUT XXX;":MACHINE1:STRACE:SEQUENCE 4,3"
```

Query Syntax: :MACHine{1|2}:STRace:SEQuence?

Returned Format: [:MACHine{1|2}:STRace:SEQuence]
<number_of_levels>,<level_of_trigger><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:SEQUENCE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

STORe

command/query

The STORe command defines the store qualifier for a given sequence level. Any data matching the STORe qualifier will actually be stored in memory as part of the current trace data. The qualifier may be a single term or a complex expression. The terms A through H are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command.

Expressions are limited to what you could manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed.

A detailed example is provided in figure 12-2.

The STORe query returns the current store qualifier specification for a given sequence level <N>.

Command Syntax: :MACHine{1|2}:STRace:STORe <N> <store_qualifier>

where:

<N> ::= an integer from 1 to the number of existing sequence levels (maximum 8)

<store_qualifier> ::= { ANYState | NOSTate | <any_term> |
(<expression1> [{AND|OR} <expression2>]) |
(<expression2> [{AND|OR} <expression1>]) }

<any_term> ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}

<expression1> ::= {<or_term1> [OR <or_term1>]... | <and_term1> [AND <and_term1>]...}

<expression2> ::= {<or_term2> [OR <or_term2>]... | <and_term2> [AND <and_term2>]...}

<or_term1> ::= {A|B|C|D|INRange|OUTRange}

<and_term1> ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}

<or_term2> ::= {E|F|G|H}

<and_term2> ::= {NOTE|NOTF|NOTG|NOTH}

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:STORE1 ANYSTATE"
OUTPUT XXX;":MACHINE1:STRACE:STORE2 OUTRANGE"
OUTPUT XXX;":MACHINE1:STRACE:STORE3 (NOTC AND NOTD AND NOTH)"
```

Query Syntax: :MACHine{1|2}:STRace:STORe<N>?

Returned Format: [:MACHine{1|2}:STRace:STORe<N>] <store_qualifier><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:STORE4?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TAG command/query

The TAG command selects the type of count tagging (state or time) to be performed during data acquisition. State tagging is indicated when the parameter is the state tag qualifier, which will be counted in the qualified state mode. The qualifier may be a single term or a complex expression. The terms A through H are defined by the TERM command. The terms INRange and OUTRange are defined by the RANGe command.

Expressions are limited to what you could manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. A detailed example is provided in figure 12-2.

Because count tagging requires a minimum clock period of 60 ns, the CPERiod and TAG commands are interrelated (the CPERiod command is in the SFORmat subsystem). When the clock period is set to Less Than, count tagging is turned off. When count tagging is set to either state or time, the clock period is automatically set to Greater Than.

The TAG query returns the current count tag specification.

Command Syntax: :MACHine{1|2}:STRace:TAG {OFF | TIME | <state_tag_qualifier>}

where:

```
<state_tag_qualifier>  ::= { ANYState | NOSTate | <any_term> |
                             (<expression1>[{AND|OR} <expression2>]) |
                             (<expression2>[{AND|OR} <expression1>]) }
<any_term>             ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
<expression1>          ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
<expression2>          ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
<or_term1>             ::= {A|B|C|D|INRange|OUTRange}
<and_term1>            ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
<or_term2>             ::= {E|F|G|H}
<and_term2>            ::= {NOTE|NOTF|NOTG|NOTH}
```

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:TAG OFF"
OUTPUT XXX;":MACHINE1:STRACE:TAG TIME"
OUTPUT XXX;":MACHINE1:STRACE:TAG (INRANGE OR NOTF)"
OUTPUT XXX;":MACHINE1:STRACE:TAG ((INRANGE OR A) AND E)"
```

Query Syntax: :MACHine{1|2} :STRace:TAG?

Returned Format: [:MACHine{1|2}:STRace:TAG] {OFF|TIME|<state_tag_qualifier>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:TAG?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TERM command/query

The TERM command allows you to a specify a pattern recognizer term in the specified machine. Each command deals with only one label in the given term; therefore, a complete specification could require several commands. Since a label can contain 32 or less bits, the range of the pattern value will be between 2^{32} - 1 and 0. When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. Since the pattern parameter may contain don't cares and be represented in several bases, it is handled as a string of characters rather than a number.

When a single state machine is on, all eight terms (A through H) are available in that machine. When two state machines are on, terms A through D are used by the first state machine defined, and terms E through H are used by the second state machine defined.

The TERM query returns the specification of the term specified by term identification and label name.

Command Syntax: :MACHine{1|2}:STRace:TERM <term_id>,<label_name>,<pattern>

where:

<term_id> ::= {A|B|C|D|E|F|G|H}

<label_name> ::= string of up to 6 alphanumeric characters

<pattern> ::= "{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"

Example:

```
OUTPUT XXX;":MACHINE1:STRACE:TERM A,'DATA','255' "
OUTPUT XXX;":MACHINE1:STRACE:TERM B,'ABC','#BXXXX1101' "
```

TERM

Query Syntax: :MACHine{1|2}:STRace:TERM? <term_id>,<label_name>

Returned Format: [:MACHine{1|2}:STRAce:TERM] <term_id>,<label_name>,<pattern><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:TERM? B,'DATA' "
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SLISt Subsystem 13

Introduction

The SLISt subsystem contains the commands available for the State Listing menu in the HP 1652B/53B logic analyzer. These commands are:

- COLumn
- DATA
- LINE
- MMODe
- OPATtern
- OSEarch
- OSTate
- OTAG
- RUNTil
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XOTag
- XPATtern
- XSEarch
- XSTate
- XTAG

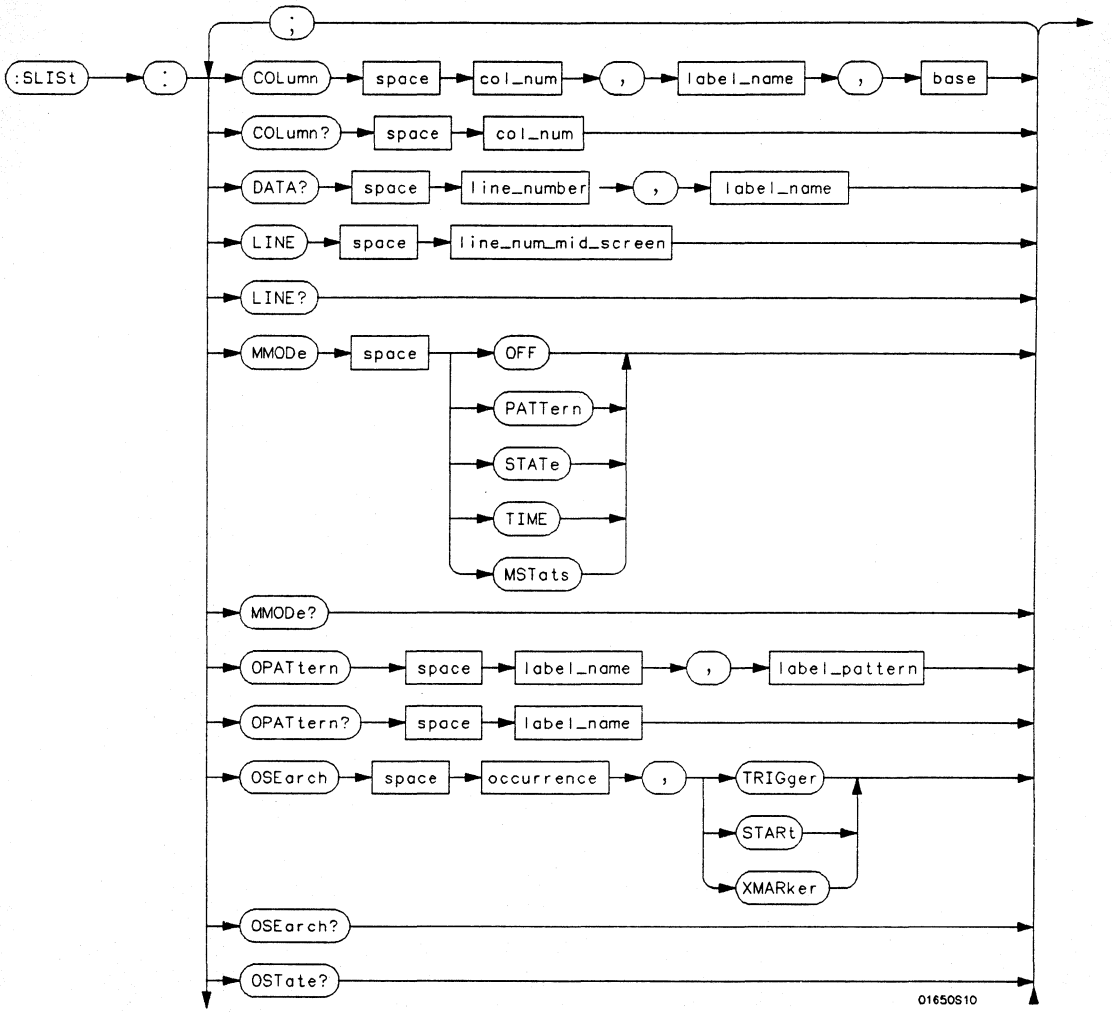


Figure 13-1. SLISt Subsystem Syntax Diagram

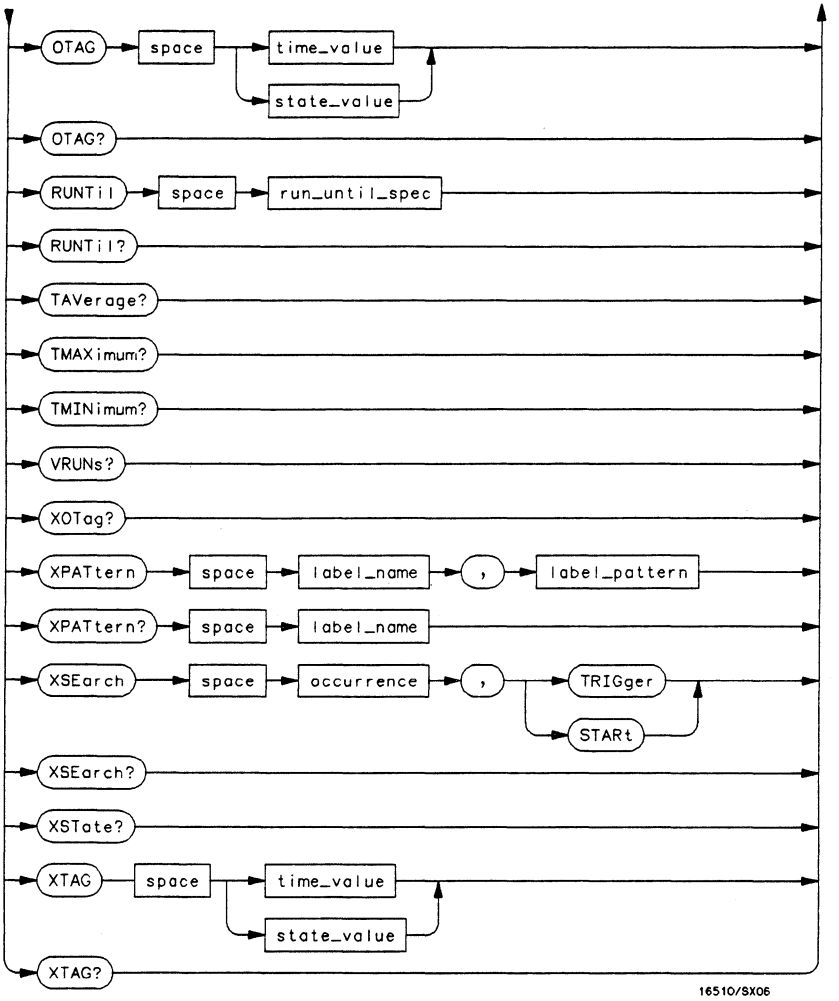


Figure 13-1. SLISt Subsystem Syntax Diagram (continued)

module_num = *{1|2|3|4|5}*
mach_num = *{1|2}*
col_num = *{1|2|3|4|5|6|7|8}*
line_number = *integer from -1023 to + 1023*
label_name = *a string of up to 6 alphanumeric characters*
base = *{BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler} for labels or*
{ABSolute|RELative} for tags
line_num_mid_screen = *integer from -1023 to + 1023*
label_pattern = "*{#B{0|1|X}... |*
#Q{0|1|2|3|4|5|6|7|X}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}... |
{0|1|2|3|4|5|6|7|8|9}... }"
occurrence = *integer from -1023 to + 1023*
time_value = *real number*
state_value = *real number*
run_until_spec = *{OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|*
OUTRange,<value>,<value>}
value = *real number*

Figure 13-1. SLISt Subsystem Syntax Diagram (continued)

SLISt selector

The SLISt selector is used as part of a compound header to access those settings normally found in the State Listing menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SLISt

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:LINE 256"
```

COLumn command/query

The COLumn command allows you to configure the state analyzer list display by assigning a label name and base to one of the eight vertical columns in the menu. A column number of 1 refers to the left most column. When a label is assigned to a column it replaces the original label in that column. The label originally in the specified column is placed in the column the specified label is moved from.

When the label name is "TAGS," the TAGS column is assumed and the next parameter must specify RELative or ABSolute.

The COLumn query returns the column number, label name, and base for the specified column.

Command Syntax: :MACHine{1|2}:SLISt:COLumn <col_num>,<label_name>,<base>

where:

<col_num> ::= {1|2|3|4|5|6|7|8}

<label_name> ::= a string of up to 6 alphanumeric characters

<base> ::= {BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler} for labels

or

::= {ABSolute|RELative} for tags

Note A label for tags must be assigned in order to use ABSolute or RELative state tagging.

Examples:

```
OUTPUT XXX;":MACHINE1:SLIST:COLUMN 4,2,MACHINE1,'A',HEX"
OUTPUT XXX;":MACHINE1:SLIST:COLUMN 1,2,MACHINE1,'TAGS', ABSOLUTE"
```

Query Syntax: :MACHine{1|2}:SLISt:COLumn? <col_num>

Returned Format: [:MACHine{1|2}:SLISt:COLumn] <col_num>,<label_name>,<base><NL>

Example:

```
10 DIM C1$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:COLUMN? 4"
30 ENTER XXX;C1$
40 PRINT C1$
50 END
```

DATA query

The DATA query returns the value at a specified line number for a given label. The format will be the same as the one shown in the Listing display except for ASCII, Symbols, or Inverse Assembly which will be returned in HEX.

Query Syntax: :MACHine{1|2}:SLISt:DATA? <line_number>,<label_name>

Returned Format: [:MACHine{1|2}:SLISt:DATA]
<line_number>,<label_name>,<pattern_string><NL>

where:

<line_number> ::= integer from -1023 to +1023

<label_name> ::= string of up to 6 alphanumeric characters

<pattern_string> ::= "{#B{0|1|X}...|
#Q{0|1|2|3|4|5|6|7|X}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
{0|1|2|3|4|5|6|7|8|9}...}"

Example:

```
10 DIM Sd$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:DATA? 512, 'RAS'"
30 ENTER XXX;Sd$
40 PRINT Sd$
50 END
```

LINE

command/query

The LINE command allows you to scroll the state analyzer listing vertically. The command specifies the state line number relative to the trigger that the analyzer will be highlighted at center screen.

The LINE query returns the line number for the state currently in the box at center screen.

Command Syntax: :MACHine{1|2}:SLISt:LINE <line_num_mid_screen>

where:

<line_num_mid_screen> ::= integer from -1023 to +1023

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:LINE 0"
```

Query Syntax: :MACHine{1|2}:SLISt:LINE?

Returned Format: [:MACHine{1|2}:SLISt:LINE] <line_num_mid_screen><NL>

Example:

```
10 DIM Ln$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:LINE?"
30 ENTER XXX;Ln$
40 PRINT Ln$
50 END
```

MMODe command/query

The MMODe command (Marker Mode) selects the mode controlling the marker movement and the display of marker readouts. When PATTern is selected, the markers will be placed on patterns. When STATe is selected and state tagging is on, the markers move on qualified states counted between normally stored states. When TIME is selected and time tagging is enabled, the markers move on time between stored states. When MSTats is selected and time tagging is on, the markers are placed on patterns, but the readouts will be time statistics.

The MMODe query returns the current marker mode selected.

Command Syntax: :MACHine{1|2}:SLISt:MMODe <marker_mode>

where:

<marker_mode> ::= {OFF|PATTern|STATe|TIME|MSTats}

Example:
```
OUTPUT XXX;":MACHINE1:SLIST:MMODE TIME"
```

Query Syntax: :MACHine{1|2}:SLISt:MMODe?

Returned Format: [:MACHine{1|2}:SLISt:MMODe] <marker_mode><NL>

Example:
```
10 DIM Mn$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:MMODE?"
30 ENTER XXX;Mn$
40 PRINT Mn$
50 END
```

OPATtern command/query

The OPATtern command allows you to construct a pattern recognizer term for the O Marker which is then used with the OSEarch criteria when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and 2^{32} - 1, since a label may not have more than 32 bits. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

The OPATtern query returns the pattern specification for a given label name.

Command Syntax: :MACHine{1|2}:SLISt:OPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<label_pattern> ::= "{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"

Examples:

```
OUTPUT XXX;":MACHINE1:SLIST:OPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:SLIST:OPATTERN 'ABC','#BXXXX1101' "
```

OPATtern

Query Syntax: :MACHine{1|2}:SLISt:OPATtern? <label_name>

Returned Format: [:MACHine{1|2}:SLISt:OPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Op$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OPATTERN? 'A'"
30 ENTER XXX;Op$
40 PRINT Op$
50 END
```

OSEarch

command/query

The OSEarch command defines the search criteria for the O marker, which is then used with associated OPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the marker to begin a search with the trigger, the start of data, or with the X marker. The actual occurrence the marker searches for is determined by the occurrence parameter of the OPATtern recognizer specification, relative to the origin. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

The OSEarch query returns the search criteria for the O marker.

Command Syntax: :MACHine{1|2}:SLISt:OSEarch <occurrence>,<origin>

where:

<occurrence> ::= integer from -1023 to +1023
<origin> ::= {TRIGger|STARt|XMARker}

Example:
```
OUTPUT XXX;":MACHINE1:SLIST:OSEARCH +10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:SLISt:OSEarch?

Returned Format: [:MACHine{1|2}:SLISt:OSEarch] <occurrence>,<origin><NL>

Example:
```
10 DIM Os$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OSEARCH?"
30 ENTER XXX;Os$
40 PRINT Os$
50 END
```

OSTate query

The OSTate query returns the line number in the listing where the O marker resides (-1023 to +1023). If data is not valid , the query returns 32767.

Query Syntax: :MACHine{1|2}:SLISt:OSTate?

Returned Format: [:MACHine{1|2}:SLISt:OSTate] <state_num><NL>

where:

<state_num> ::= an integer from -1023 to +1023, or 32767

Example:

```
10 DIM Os$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OSTATE?"
30 ENTER XXX;Os$
40 PRINT Os$
50 END
```

OTAG command/query

The OTAG command specifies the tag value on which the O Marker should be placed. The tag value is time when time tagging is on or states when state tagging is on. If the data is not valid tagged data, no action is performed.

The OTAG query returns the O Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time or through a pattern search. If data is not valid, the query returns 9.9E37 for time tagging, 32767 for state tagging.

Command Syntax: :MACHine{1|2}:SLISt:OTAG {<time_value>|<state_value>}

where:

<time_value> ::= real number
<state_value> ::= integer

Example:

```
:OUTPUT XXX;":MACHINE1:SLIST:OTAG 40.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:OTAG?

Returned Format: [:MACHine{1|2}:SLISt:OTAG] {<time_value>|<state_value>}<NL>

Example:

```
10 DIM Ot$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OTAG?"
30 ENTER XXX;Ot$
40 PRINT Ot$
50 END
```

RUNTil command/query

The RUNTil (run until) command allows you to define a stop condition when the trace mode is repetitive. Specifying OFF causes the analyzer to make runs until either the display's STOP field is touched or the STOP command is issued.

There are four conditions based on the time between the X and O markers. Using this difference in the condition is effective only when time tags have been turned on (see the TAG command in the STRace subsystem). These four conditions are as follows:

- The difference is less than (LT) some value.
- The difference is greater than (GT) some value.
- The difference is inside some range (INRange).
- The difference is outside some range (OUTRange).

End points for the INRange and OUTRange should be at least 40 ns apart since this is the minimum time resolution of the time tag counter.

There are two conditions which are based on a comparison of the acquired state data and the compare data image. You can run until one of the following conditions is true:

- Compare Equal (EQUal) - Every channel of every label has the same value.
- Compare not equal (NEQual) - Any channel of any label has a different value.

The RUNTil query returns the current stop criteria.

Note 

The RUNTil instruction (for state analysis) is available in both the SLISt and COMPare subsystems.

Command Syntax: :MACHine{1|2}:SLISt:RUNTil <run_until_spec>

where:

<run_until_spec> ::= {OFF|LT,<value>|GT,<value>|INRange,<value>,<value>
|OUTRange,<value>,<value>|EQUal|NEQual}

<value> ::= real number from 10E-9 to +9E9

Example:
```
OUTPUT XXX;":MACHINE1:SLIST:RUNTIL GT,800.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:RUNTil?

Returned Format: [:MACHine{1|2}:SLISt:RUNTil] <run_until_spec><NL>

Example:
```
10 DIM Ru$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:RUNTIL?"
30 ENTER XXX;Ru$
40 PRINT Ru$
50 END
```

TAVerage query

The TAVerage query returns the value of the average time between the X and O Markers. If the number of valid runs is zero, the query returns 9.9E37. Valid runs are those where the pattern search for both the X and O markers was successful, resulting in valid delta-time measurements.

Query Syntax: :MACHine{1|2}:SLISt:TAVerage?

Returned Format: [:MACHine{1|2}:SLISt:TAVerage] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tv$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TAVERAGE?"
30 ENTER XXX;Tv$
40 PRINT Tv$
50 END
```

TMAXimum query

The TMAXimum query returns the value of the maximum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:SLISt:TMAXimum?

Returned Format: [:MACHine{1|2}:SLISt:TMAXimum] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tx$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TMAXIMUM?"
30 ENTER XXX;Tx$
40 PRINT Tx$
50 END
```

TMINimum query

The TMINimum query returns the value of the minimum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:SLISt:TMINimum?

Returned Format: [:MACHine{1|2}:SLISt:TMINimum] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tm$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TMINIMUM?"
30 ENTER XXX;Tm$
40 PRINT Tm$
50 END
```

VRUNs

query

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Query Syntax: :MACHine{1|2}:SLISt:VRUNs?

Returned Format: [:MACHine{1|2}:SLISt:VRUNs] <valid_runs>,<total_runs><NL>

where:

<valid_runs> ::= zero or positive integer
<total_runs> ::= zero or positive integer

Example:

```
10 DIM Vr$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:VRUNS?"
30 ENTER XXX;Vr$
40 PRINT Vr$
50 END
```

XOTag query

The XOTag query returns the time from the X to O markers when the marker mode is time or number of states from the X to O markers when the marker mode is state. If there is no data in the time mode the query returns 9.9E37. If there is no data in the state mode, the query returns 32767.

Query Syntax: :MACHine{1|2}:SLISt:XOTag?

Returned Format: [:MACHine{1|2}:SLISt:XOTag] {<XO_time>|<XO_states>}<NL>

where:

<XO_time> ::= real number
<XO_states> ::= integer

Example:

```
10 DIM Xot$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XOTAG?"
30 ENTER XXX;Xot$
40 PRINT Xot$
50 END
```

XPATtern command/query

The XPATtern command allows you to construct a pattern recognizer term for the X Marker which is then used with the XSEarch criteria when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and 2^{32} - 1, since a label may not have more than 32 bits. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

The XPATtern query returns the pattern specification for a given label name.

Command Syntax: :MACHine{1|2}:SLISt:XPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<label_pattern> ::= "{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"

Examples:

```
OUTPUT XXX;":MACHINE1:SLIST:XPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:SLIST:XPATTERN 'ABC','#BXXXX1101' "
```

XPATtern

Query Syntax: :MACHine{1|2}:SLISt:XPATtern? <label_name>

Returned Format: [:MACHine{1|2}:SLISt:XPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Xp$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XPATTERN? 'A'"
30 ENTER XXX;Xp$
40 PRINT Xp$
50 END
```

XSEarch

command/query

The XSEarch command defines the search criteria for the X Marker, which is then with associated XPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the Marker to begin a search with the trigger or with the start of data. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 places a marker on the selected origin.

The XSEarch query returns the search criteria for the X marker.

Command Syntax: :MACHine{1|2}:SLISt:XSEarch <occurrence>,<origin>

where:

<occurrence> ::= integer from -1023 to +1023
<origin> ::= {TRIGger|STARt}

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:XSEARCH +10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:SLISt:XSEarch?

Returned Format: [:MACHine{1|2}:SLISt:XSEarch] <occurrence>,<origin><NL>

Example:

```
10 DIM Xs$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XSEARCH?"
30 ENTER XXX;Xs$
40 PRINT Xs$
50 END
```

XSTate query

The XSTate query returns the line number in the listing where the X marker resides (-1023 to +1023). If data is not valid, the query returns 32767.

Query Syntax: :MACHine{1|2}:SLISt:XSTate?

Returned Format: [:MACHine{1|2}:SLISt:XSTate] <state_num><NL>

where:

<state_num> ::= an integer from -1023 to +1023, or 32767

Example:

```
10 DIM Xs$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XSTATE?"
30 ENTER XXX;Xs$
40 PRINT Xs$
50 END
```

XTAG command/query

The XTAG command specifies the tag value on which the X Marker should be placed. The tag value is time when time tagging is on or states when state tagging is on. If the data is not valid tagged data, no action is performed.

The XTAG query returns the X Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time or through a pattern search. If data is not valid tagged data, the query returns 9.9E37 for time tagging, 32767 for state tagging.

Command Syntax: :MACHine{1|2}:SLISt:XTAG {<time_value> | <state_value>}

where:

<time_value> ::= real number
<state_value> ::= integer

Example:

```
:OUTPUT XXX;":MACHINE1:SLIST:XTAG 40.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:XTAG?

Returned Format: [:MACHine{1|2}:SLISt:XTAG] {<time_value> | <state_value>}<NL>

Example:

```
10 DIM Xt$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XTAG?"
30 ENTER XXX;Xt$
40 PRINT Xt$
50 END
```

SWAVeform Subsystem 14

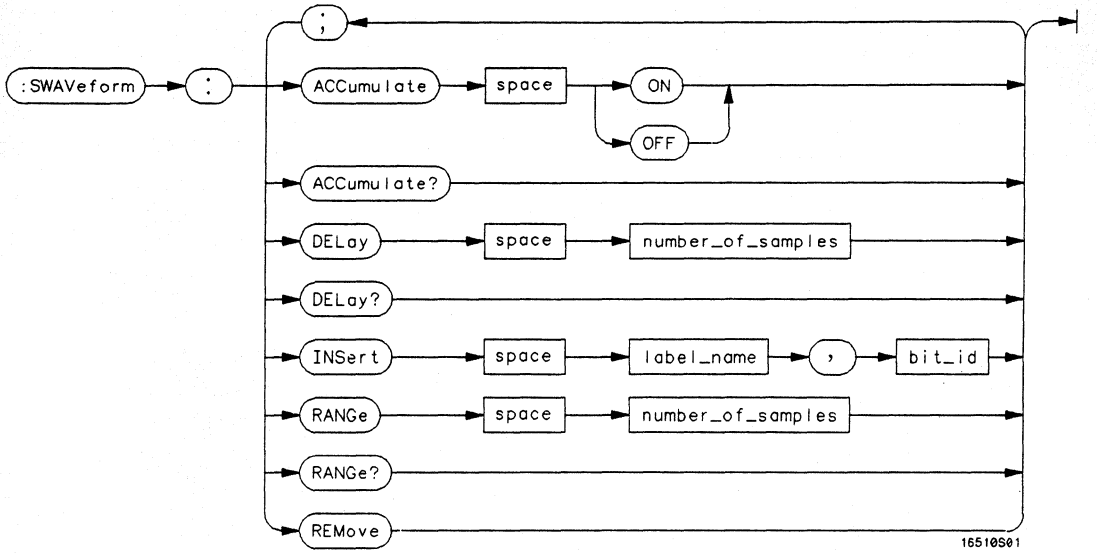
Introduction

The commands in the State Waveform subsystem allow you to configure the display so that you can view state data as waveforms on up to 24 channels identified by label name and bit number. The five commands are analogous to their counterparts in the Timing Waveform subsystem. However, in this subsystem the x-axis is restricted to representing only samples (states), regardless of whether time tagging is on or off. As a result, the only commands which can be used for scaling are DELay and RANge.

The way to manipulate the X and O markers on the Waveform display is through the State Listing (SLISt) subsystem. Using the marker commands from the SLISt subsystem will affect the markers on the Waveform display.

The commands in the SWAVeform subsystem are:

- ACCumulate
- DELay
- INSert
- RANGe
- REMove



number_of_samples = *integer from -1023 to + 1024*
label_name = *string of up to 6 alphanumeric characters*
bit_id = {*OVERlay* | <*bit_num*>}
bit_num = *integer representing a label bit from 0 to 31*

Figure 14-1. SWAVeform Subsystem Syntax Diagram

SWAVeform selector

The SWAVeform (State Waveform) selector is used as part of a compound header to access the settings in the State Waveform menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SWAVeform

Example:

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE 40"
```

ACCumulate command/query

The ACCumulate command allows you to control whether the waveform display gets erased between individual runs or whether subsequent waveforms are allowed to be displayed over the previous waveforms.

The ACCumulate query returns the current setting. The query always shows the setting as the character "0" (off) or "1" (on).

Command Syntax: :MACHine{1|2}:SWAVeform:ACCumulate {{ON | 1} | {OFF | 0}}

Example:
```
OUTPUT XXX;":MACHINE1:SWAVEFORM:ACCUMULATE ON"
```

Query Syntax: MACHine{1|2}:SWAVeform:ACCumulate?

Returned Format: [MACHine{1|2}:SWAVeform:ACCumulate] {0 | 1}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SWAVEFORM:ACCUMULATE?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

DELay command/query

The DELay command allows you to specify the number of samples between the timing trigger and the horizontal center of the screen for the waveform display. The allowed number of samples is from -1023 to +1024.

The DELay query returns the current sample offset value.

Command Syntax: :MACHine{1|2}:SWAVeform:DELay <number_of_samples>

where:

<number_of_samples> ::= integer from −1023 to +1024

Example:
```
OUTPUT XXX;":MACHINE2:SWAVEFORM:DELAY 127"
```

Query Syntax: MACHine{1|2}:SWAVeform:DELay?

Returned Format: [MACHine{1|2}:SWAVeform:DELay] <number_of_samples><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SWAVEFORM:DELAY?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

INSert command

The INSert command allows you to add waveforms to the state waveform display. Waveforms are added from top to bottom on the screen. When 24 waveforms are present, inserting additional waveforms replaces the last waveform. Bit numbers are zero based, so a label with 8 bits is referenced as bits 0-7. Specifying OVERlay causes a composite waveform display of all bits or channels for the specified label.

Command Syntax: MACHine{1|2}:SWAVeform:INSert <label_name>,<bit_id>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<bit_id> ::= {OVERlay|<bit_num>}
<bit_num> ::= integer representing a label bit from 0 to 31

Examples:

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:INSERT 'WAVE', 19"
OUTPUT XXX;":MACHINE1:SWAVEFORM:INSERT 'ABC', OVERLAY"
OUTPUT XXX;":MACH1:SWAV:INSERT 'POD1', #B1001"
```

RANGe command/query

The RANGe command allows you to specify the number of samples across the screen on the State Waveform display. It is equivalent to ten times the states per division setting (st/Div) on the front panel. A number between 10 and 10240 may be entered.

The RANGe query returns the current range value.

Command Syntax: MACHine{1|2}:SWAVeform:RANGe <number_of_samples>

where:

<number_of_samples> ::= integer from 10 to 10240

Example:

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE 80"
```

Query Syntax: MACHine{1|2}:SWAVeform:RANGe?

Returned Format: [MACHine{1|2}:SWAVeform:RANGe] <number_of_samples><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

REMove command

The REMove command allows you to clear the waveform display before building a new display.

Command Syntax: :MACHine{1|2}:SWAVeform:REMove

Example:

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:REMOVE"
```

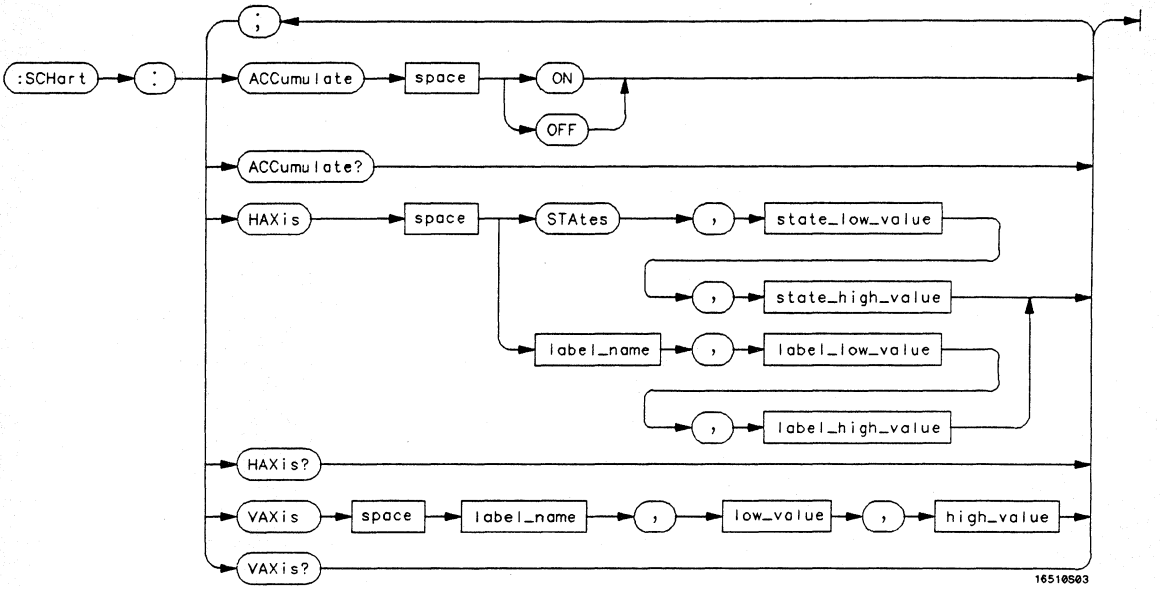
SCHart Subsystem 15

Introduction

The State Chart subsystem provides the commands necessary for programming the HP 1652B/53B's Chart display. The commands allow you to build charts of label activity, using data normally found in the Listing display. The chart's y-axis is used to show data values for the label of your choice. The x-axis can be used in two different ways. In one, the x-axis represents states (shown as rows in the State Listing display). In the other, the x-axis represents the data values for another label. When states are plotted along the x-axis, X and O markers are available. Since the State Chart display is simply an alternative way of looking at the data in the State Listing, the X and O markers can be manipulated through the SLISt subsystem. In fact, because the programming commands do not force the menus to switch, you can position the markers in the SLISt subsystem and see the effects in the State Chart display.

The commands in the SCHart subsystem are:

- ACCumulate
- HAXis
- VAXis



state_low_value = *integer from –1023 to + 1024*
state_high_value = *integer from <state_low_value> to + 1024*
label_name = *a string of up to 6 alphanumeric characters*
label_low_value = *string from 0 to* 2^{32} *- 1 (#HFFFFFFFF)*
label_high_value = *string from <label_low_value> to* 2^{32} *- 1 (#HFFFFFFFF)*
low_value = *string from 0 to* 2^{32} *- 1 (#HFFFFFFFF)*
high_value = *string from < low_value> to* 2^{32} *- 1 (#HFFFFFFFF)*

Figure 15-1. SCHart Subsystem Syntax Diagram

SCHart selector

The SCHart selector is used as part of a compound header to access the settings found in the State Chart menu. It always follows the MACHine selector because it selects a branch below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SCHart

Example:

```
OUTPUT XXX;":MACHINE1:SCHART:VAXIS 'A', '0', '9'"
```

ACCumulate command/query

The ACCumulate command allows you to control whether the chart display gets erased between each individual run or whether subsequent waveforms are allowed to be displayed over the previous waveforms.

The ACCumulate query returns the current setting. The query always shows the setting as the character "0" (off) or "1" (on).

Command Syntax: MACHine{1|2}:SCHart:ACCumulate {{ON | 1} | {OFF | 0}}

Example:
```
OUTPUT XXX;":MACHINE1:SCHART:ACCUMULATE OFF"
```

Query Syntax: MACHine{1|2}:SCHart:ACCumulate?

Returned Format: [MACHine{1|2}:SCHart:ACCumulate] {0 | 1}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SCHART:ACCUMULATE?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

HAXis

command/query

The HAXis command allows you to select whether states or a label's values will be plotted on the horizontal axis of the chart. The axis is scaled by specifying the high and low values.

Note

The shortform for STATES is STA. This is an intentional deviation from the normal trunctation rules.

The HAXis query returns the current horizontal axis label assignment and scaling.

Command Syntax: MACHine{1|2}:SCHart:HAXis {STATES,<state_low_value>,<state_high_value> | <label_name>,<label_low_value>,<label_high_value>}

where:

<state_low_value> ::= integer from -1023 to 1024
<state_high_value> ::= integer from <state_low_value> to +1024
<label_name> ::= a string of up to 6 alphanumeric characters
<label_low_value> ::= string from 0 to 2^{32}-1 (#HFFFFFFFF)
<label_high_value> ::= string from <label_low_value> to 2^{32}-1 (#HFFFFFFFF)

Examples:

```
OUTPUT XXX;":MACHINE1:SCHART:HAXIS STATES, -100, 100"
OUTPUT XXX;":MACHINE1:SCHART:HAXIS 'DATA', '100', '511'"
```

HAXis

Query Syntax: MACHine{1|2}:SCHart:HAXis?

Returned Format: [MACHine{1|2}:SCHart:HAXis] {STATES,<state_low_value>,<state_high_value> | <label_name>,<label_low_value>,<label_high_value>}

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SCHART:HAXIS?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

VAXis command/query

The VAXis command allows you to choose which label will be plotted on the vertical axis of the chart and scale the vertical axis by specifying the high value and low value.

The VAXis query returns the current vertical axis label assignment and scaling.

Command Syntax: MACHine{1|2}:SCHart:VAXis <label_name>,<low_value>,<high_value>

where:

<label_name> ::= a string of up to 6 alphanumeric characters
<low_value> ::= string from 0 to 2^{32}-1 (#HFFFFFFFF)
<high_value> ::= string from <low_value> to 2^{32}-1 (#HFFFFFFFF)

Examples:

```
OUTPUT XXX;":MACHINE2:SCHART:VAXIS 'SUM1', '0', '99'"
OUTPUT XXX;":MACHINE1:SCHART:VAXIS 'BUS', '#H00FF', '#H0500'"
```

Query Syntax: MACHine{1|2}:SCHart:VAXis?

Returned Format: [MACHine{1|2}:SCHart:VAXis] <label_name>,<low_value>,<high_value><NL>

Example:

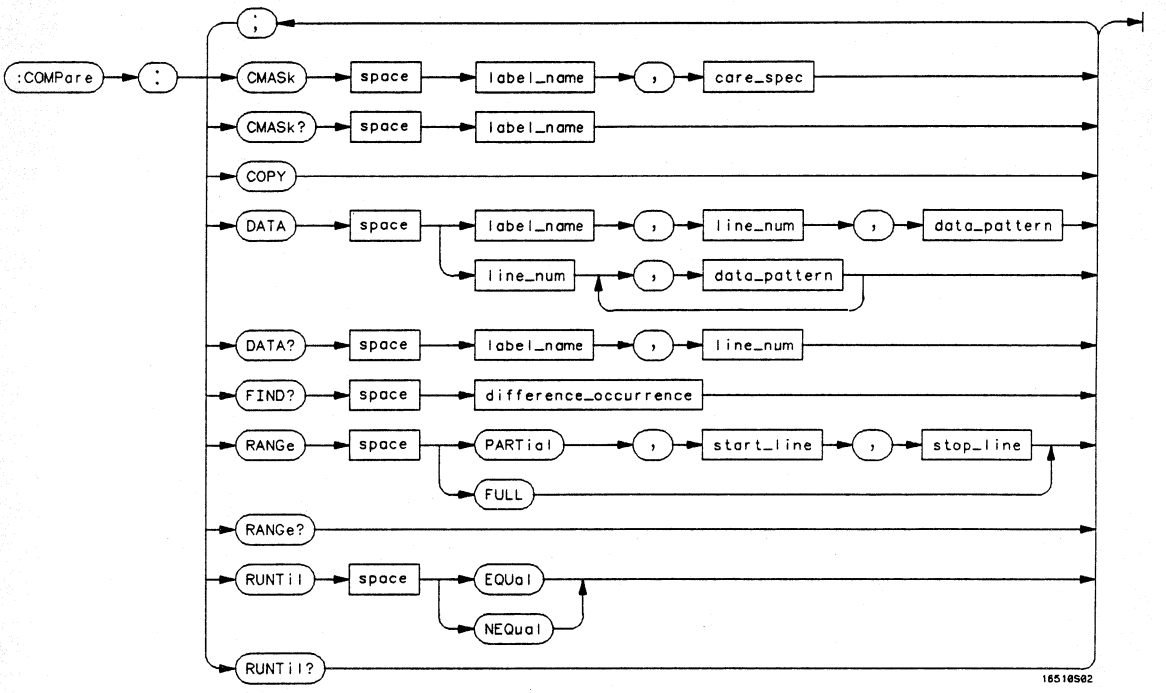
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SCHART:VAXIS?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

COMPare Subsystem 16

Introduction

Commands in the state COMPare subsystem provide the ability to do a bit-by-bit comparison between the acquired state data listing and a compare data image. The commands are:

- COPY
- DATA
- CMASk
- RANGe
- RUNTil
- FIND



label_name = *string of up to 6 characters*
care_spec = *string of characters* "{*|.}..."
* = *care*
. = *don't care*
line_num = *integer from −1023 to +1023*
data_pattern = "{#B{0|1|X}... |
#Q{0|1|2|3|4|5|6|7|X}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}... |
{0|1|2|3|4|5|6|7|8|9}... }"
difference_occurence = *integer from 1 to 1024*
start_line = *integer from −1023 to +1023*
stop_line = *integer from <start_line> to +1023*

Figure 16-1. COMPare Subsystem Syntax Diagram

COMPare selector

The COMPare selector is used as part of a compound header to access the settings found in the Compare menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:COMPare

Example:

```
OUTPUT XXX;":MACHINE1:COMPARE:FIND? 819"
```

CMASk command/query

The CMASk (Compare Mask) command allows you to set the bits in the channel mask for a given label in the compare listing image to "compares" or "don't compares."

The CMASk query returns the state of the bits in the channel mask for a given label in the compare listing image.

Command Syntax: MACHine{1|2}:COMPare:CMASk <label_name>,<care_spec>

where:

<label_name> ::= a string of up to 6 alphanumeric characters
<care_spec> ::= string of characters "{*|.}..." (32 characters maximum)
* ::= care
::= don't care

Example:

```
OUTPUT XXX;":MACHINE2:COMPARE:CMASK 'STAT', '*.**..**'"
```

Query Syntax: MACHine{1|2}:COMPare:CMASk? <label_name>

Returned Format: [MACHine{1|2}:COMPare:CMASk] <label_name>,<care_spec><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:COMPARE:CMASK? 'POD5'"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

COPY command

The COPY command copies the current acquired State Listing for the specified machine into the Compare Listing template. It does not affect the compare range or channel mask settings.

Command Syntax: MACHine{1|2}:COMPare:COPY

Example:

```
OUTPUT XXX;":MACHINE2:COMPARE:COPY"
```

DATA command/query

The DATA command allows you to edit the compare listing image for a given label and state row. When DATA is sent to an instrument where no compare image is defined (such as at power-up) all other data in the image is set to don't cares.

Not specifying the <label_name> parameter allows you to write data patterns to more than one label for the given line number. The first pattern is placed in the left-most label, with the following patterns being placed in a left-to-right fashion (as seen on the Compare display). Specifying more patterns than there are labels simply results in the extra patterns being ignored.

Because don't cares (Xs) are allowed in the data pattern, it must always be expressed as a string. You may still use different bases, though don't cares cannot be used in a decimal number.

The DATA query returns the value of the compare listing image for a given label and state row.

Command Syntax: MACHine{1|2}:COMPare:DATA {<label_name>,<line_num>,<data_pattern> |
<line_num>,<data_pattern>[, <data_pattern>]... }

where:

<label_name> ::= a string of up 6 alphanumeric characters
<line_num> ::= integer from −1023 to +1023
<data_pattern> ::= "{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"

Examples:

```
OUTPUT XXX;":MACHINE2:COMPARE:DATA 'CLOCK', 42, '#B011X101X'"
OUTPUT XXX;":MACHINE2:COMPARE:DATA 'OUT3', 0, '#HFF40'"
OUTPUT XXX;":MACHINE1:COMPARE:DATA 129, '#BXX00', '#B1101', '#B10XX'"
OUTPUT XXX;":MACH2:COMPARE:DATA -511, '4', '64', '16', 256', '8', '16'"
```

Query Syntax: MACHine{1|2}:COMPare:DATA? <label_name>,<line_num>

Returned Format: [MACHine{1|2}:COMPare:DATA]
<label_name>,<line_num>,<data_pattern><NL>

Example:

```
10 DIM Label$[6], Response$[80]
15 PRINT "This program shows the values for a signal's Compare listing"
20 INPUT "Enter signal label: ", Label$
25 OUTPUT XXX;":SYSTEM:HEADER OFF"    !Turn headers off (from responses)
30 OUTPUT XXX;":MACHINE2:COMPARE:RANGE?"
35 ENTER XXX; First, Last                !Read in the range's end-points
40 PRINT "LINE #", "VALUE of "; Label$
45 FOR State = First TO Last         !Print compare value for each state
50   OUTPUT XXX;":MACH2:COMPARE:DATA? '" & Label$ & "'," & VAL$(State)
55   ENTER XXX; Response$
60   PRINT State, Response$
65   NEXT State
70 END
```

FIND query

The FIND query is used to get the line number of a specified difference occurence (first, second, third, etc) within the current compare range, as dictated by the RANGe command (see RANGe). A difference is counted for each line where at least one of the current labels has a discrepancy between its acquired state data listing and its compare data image.

Invoking the FIND query updates both the Listing and Compare displays so that the line number returned is in the center of the screen.

Query Syntax: MACHine{1|2}:COMPare:FIND? <difference_occurrence>

Returned Format: [MACHine{1|2}:COMPare:FIND] <difference_occurrence>, <line_number><NL>

where:

<difference_occurrence> ::= integer from 0 to 1024

<line_number> ::= integer from −1023 to +1023

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:COMPARE:FIND? 26"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

RANGe command/query

The RANGe command allows you to define the boundaries for the comparison. The range entered must be a subset of the lines in the aquisition memory.

The RANGe query returns the current boundaries for the comparison.

Command Syntax: MACHine{1|2}:COMPare:RANGe {FULL | PARTial,<start_line>,<stop_line>}

where:

<start_line> ::= integer from −1023 to +1023
<stop_line> ::= integer from <start_line> to +1023

Examples:

```
OUTPUT XXX;":MACHINE2:COMPARE:RANGE PARTIAL, -511, 512"
OUTPUT XXX;":MACHINE2:COMPARE:RANGE FULL"
```

Query Syntax: MACHine{1|2}:COMPare:RANGe?

Returned Format: [MACHine{1|2}:COMPare:RANGe] {FULL | PARTial,<start_line>, <stop_line>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE4:COMPARE:RANGE?"
30 ENTER XXX; String$
40 REM  See if substring "FULL" occurs in response string:
50 PRINT "Range is ";
60 IF POS(String$,"FULL") > 0 THEN PRINT "Full" ELSE PRINT "Partial"
70 END
```

RUNTil command/query

The RUNTil (run until) command allows you to define a stop condition when the trace mode is repetitive. Specifying OFF causes the analyzer to make runs until either the display's STOP field is touched or the STOP command is issued.

There are four conditions based on the time between the X and O markers. Using this difference in the condition is effective only when time tags have been turned on (see the TAG command in the STRace subsystem). These four conditions are as follows:

- The difference is less than (LT) some value.
- The difference is greater than (GT) some value.
- The difference is inside some range (INRange).
- The difference is outside some range (OUTRange).

End points for the INRange and OUTRange should be at least 40 ns apart.

There are two conditions which are based on a comparison of the acquired state data and the compare data image. You can run until one of the following conditions is true:

- Compare equal (EQUal) - Every channel of every label has the same value.
- Compare not equal (NEQual) - Any channel of any label has a different value .

The RUNTil query returns the current stop criteria for the comparison when running in repetitive trace mode.

Note 

The RUNTil instruction (for state analysis) is available in both the SLISt and COMPare subsystems.

Command Syntax: MACHine{1|2}:COMPare:RUNTil {OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual}

Example:
```
OUTPUT XXX;":MACHINE2:COMPARE:RUNTIL EQUAL"
```

Query Syntax: MACHine{1|2}:COMPare:RUNTil?

Returned Format: [MACHine{1|2}:COMPare:RUNTil] {OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual}<NL>

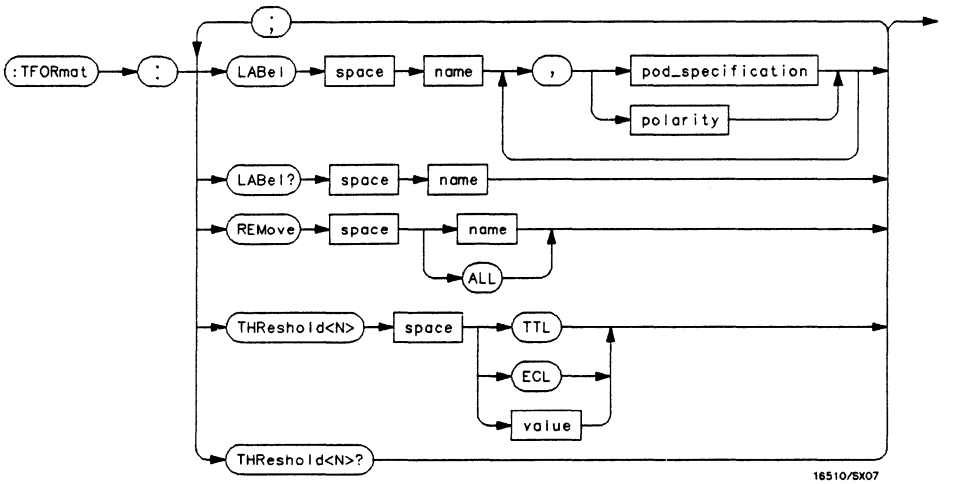
Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:COMPARE:RUNTIL?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

TFORmat Subsystem 17

Introduction

The TFORmat subsystem contains the commands available for the Timing Format menu in the HP 1652B/53B logic analyzer. These commands are:

- LABel
- REMove
- THReshold



<N> = {*1* | *2* | *3* | *4* | *5*}
name = *string of up to 6 alphanumeric characters*
polarity = {*POSitive* | *NEGative*}
pod_specification = *format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)*
value = *voltage (real number) -9.9 to +9.9*

Figure 17-1. TFORmat Subsystem Syntax Diagram

TFORmat selector

The TFORmat selector is used as part of a compound header to access those settings normally found in the Timing Format menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the language tree.

Command Syntax: :MACHine{1|2}:TFORmat

Example:

```
OUTPUT XXX;":MACHINE1:TFORMAT:LABEL?"
```

LABel

command/query

The LABel command allows you to specify polarity and assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label will be created.

The order of the pod-specification parameters is significant. The first one listed will match the highest-numbered pod assigned to the machine you're using. Each pod specification after that is assigned to the next-highest-numbered pod. This way they match the left-to-right descending order of the pods you see on the Format display. Not including enough pod specifications results in the lowest-numbered pod(s) being assigned a value of zero (all channels excluded). If you include more pod specifications than there are pods for that machine, the extra ones will be ignored. However, an error is reported anytime more than five pod specifications are listed.

The polarity can be specified at any point after the label name.

Since pods contain 16 channels, the format value for a pod must be between 0 and 65535 (2^{16}-1). When giving the pod assignment in binary (base 2), each bit will correspond to a single channel. A "1" in a bit position means the associated channel in that pod is assigned to that pod and bit. A "0" in a bit position means the associated channel in that pod is excluded from the label. For example, assigning #B1111001100 is equivalent to entering "....****..**.." through the front-panel user interface.

A label can not have a total of more than 32 channels assigned to it.

The LABel query returns the current specification for the selected (by name) label. If the label does not exist, nothing is returned. Numbers are always returned in decimal format.

LABel

Command Syntax: :MACHine{1|2}:TFORmat:LABel <name>[,{<polarity> | <assignment>}]...

where:

<name> ::= string of up to 6 alphanumeric characters
<polarity> ::= {POSitive | NEGative}
<assignment> ::= format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

Examples:

```
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'DATA', POS, 65535, 127, 40312"
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'STAT', 1, 8096, POSITIVE"
OUTPUT XXX;":MACHINE1:TFORMAT:LABEL 'ADDR', NEGATIVE, #B11110010101010"
```

Query Syntax: :MACHine{1|2}:TFORmat:LABel? <name>

Returned Format: [:MACHine{1|2}:TFORmat:LABel] <name>[,<assignment>]...,<polarity><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:TFORMAT:LABEL? 'DATA'"
30 ENTER XXX String$
40 PRINT String$
50 END
```

REMove command

The REMove command allows you to delete all labels or any one label specified by name for a given machine.

Command Syntax: :MACHine{1|2}:TFORmat:REMove {<name>|ALL}

where:

<name> ::= string of up to 6 alphanumeric characters

Examples:

```
OUTPUT XXX;":MACHINE1:TFORMAT:REMOVE 'A'"
OUTPUT XXX;":MACHINE1:TFORMAT:REMOVE ALL"
```

THReshold command/query

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL or a specific voltage from -9.9V to +9.9V in 0.1 volt increments.

Note 

On the HP 1652B, the pod thresholds of pods 1, 2, and 3 can be set independently. The pod thresholds of pods 4 and 5 are slaved together; therefore, when you set the threshold on pod 4 or 5, both thresholds will be changed to the specified value. On the HP 1653B, both pods 1 and 2 can be set independently.

The THReshold query returns the current threshold for a given pod.

Command Syntax: :MACHine{1|2}:TFORmat:THReshold<N> {TTL|ECL|<value>}

where:

<N> ::= pod number {1|2|3|4|5}
<value> ::= voltage (real number) -9.9 to +9.9
TTL ::= default value of +1.6V
ECL ::= default value of -1.3V

Example:
```
OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD1 4.0"
```

Query Syntax: :MACHine{1|2}:TFORmat:THReshold<N>?

Returned Format: [:MACHine{1|2}:TFORmat:THReshold<N>] <value><NL>

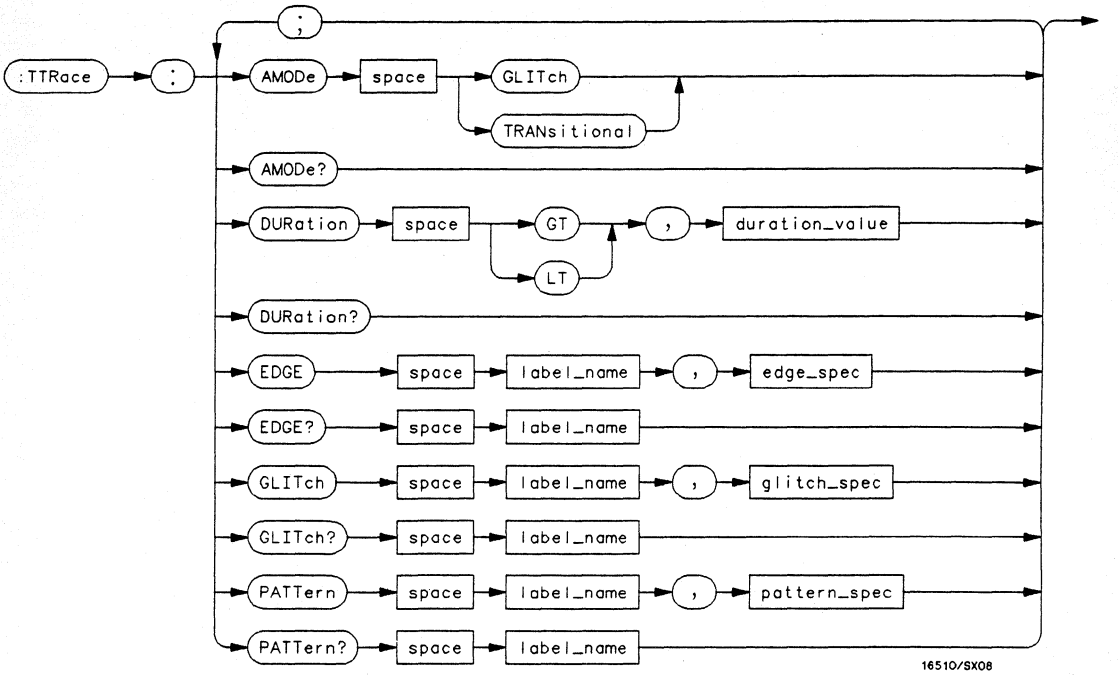
Example:
```
10 DIM Value$ [100]
20 OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD2?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

TTRace Subsystem 18

Introduction

The TTRace subsystem contains the commands available for the Timing Trace menu in the HP 1652B/53B logic analyzer. These commands are:

- AMODe
- DURation
- EDGE
- GLITch
- PATTern



GT = *greater than*
LT = *less than*
duration_value = *real number*
label_name = *string of up to 6 alphanumeric characters*
edge_spec = *string of characters* "{*R*|*F*|*T*|*X*}..."
R = *rising edge*
F = *falling edge*
T = *toggling or either edge*
X = *don't care or ignore this channel*
glitch_spec = *string of characters* "{*|.}..."
***** = *search for a glitch on this channel*
. = *ignore this channel*
pattern_spec = "{#*B*{*0*|*1*|*X*}... |
#*Q*{*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*X*}... |
#*H*{*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*|*A*|*B*|*C*|*D*|*E*|*F*|*X*}... |
{*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*}... }"

Figure 18-1. TTRace Subsystem Syntax Diagram

TTRace selector

The TTRace selector is used as part of a compound header to access the settings found in the Timing Trace menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the language tree.

Command Syntax: :MACHine{1|2}:TTRace

Example:

```
OUTPUT XXX;":MACHINE1:TTRACE:GLITCH 'ABC', '....****'"
```

AMODe command/query

The AMODe command allows you to select the acquisition mode used for a particular timing trace. The acquisition modes available are TRANsitional and GLITch.

The AMODe query returns the current acquisition mode.

Command Syntax: :MACHine{1|2}:TTRace:AMODe <acquisition_mode>

where:

<acquisition_mode> ::= {GLITch|TRANsitional}

Example:
```
OUTPUT XXX; ":MACHINE1:TTRACE:AMODE GLITCH"
```

Query Syntax: :MACHine1:TTRace:AMODe?

Returned Format: [:MACHine1:TTRace:AMODe] {GLITCH|TRANSITIONAL}

Example:
```
10 DIM M$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:AMODE?"
30 ENTER XXX;M$
40 PRINT M$
50 END
```

DURation command/query

The DURation command allows you to specify the duration qualifier to be used with the pattern recognizer term in generating the timing trigger. The duration value can be specified in 10 ns increments within the following ranges:

- Greater than (GT) qualification - 30 ns to 10 ms
- Less than (LT) qualification - 40 ns to 10 ms.

The DURation query returns the current pattern duration qualifier specification.

Command Syntax: :MACHine{1|2}:TTRace:DURation {GT|LT},<duration_value>

where:

GT ::= greater than
LT ::= less than
<duration_value> ::= real number

Example:
```
OUTPUT XXX; ":MACHINE1:TTRACE:DURATION GT, 40.0E-9"
```

Query Syntax: :MACHine{1|2}:TTRace:DURation?

Returned Format: [:MACHine{1|2}:TTRace:DURation] {GT|LT},<duration_value><NL>

Example:
```
10 DIM D$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:DURATION?"
30 ENTER XXX;D$
40 PRINT D$
50 END
```

EDGE command/query

The EDGE command allows you to specify the edge recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in the given edge specification; therefore, a complete specification could require several commands. The edge specification uses the characters R, F, T, X to indicate the edges or don't cares as follows:

R = rising edge
F = falling edge
T = toggling or either edge
X = don't care or ignore the channel

The position of these characters in the string corresponds with the position of the channels within the label. All channels without "X" are ORed together to form the edge trigger specification.

The EDGE query returns the edge specification for the specified label.

Command Syntax: :MACHine{1|2}:TTRace:EDGE <label_name>,<edge_spec>

where:

<label_name> ::= string or up to 6 alphanumeric characters
<edge_spec> ::= string of characters "{R|F|T|X}..."

Example:

```
OUTPUT XXX; ":MACHINE1:TTRACE:EDGE 'POD1','XXXXXXXR'"
```

Query Syntax: :MACHine{1|2}:TTRace:EDGE? <label_name>

Returned Format: [:MACHine{1|2}:TTRace:] <label_name>,<edge_spec><NL>

Example:

```
10 DIM E$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:EDGE? 'POD1'"
30 ENTER XXX;E$
40 PRINT E$
50 END
```

GLITch command/query

The GLITch command allows you to specify the glitch recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in a given glitch specification, and, therefore a complete specification could require several commands. The glitch specification uses the characters "*" and "." as follows:

"*" (asterisk) = search for a glitch on this channel

"." (period) = ignore this channel

The position of these characters in the string corresponds with the position of the channels within the label. All channels with the "*" are ORed together to form the glitch trigger specification.

The GLITch query returns the glitch specification for the specified label.

Command Syntax: :MACHine{1|2}:TTRace:GLITch <label_name>,<glitch_spec>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<glitch_spec> ::= string of characters "{*|.}..."

Example:
```
OUTPUT XXX; ":MACHINE1:TTRACE:GLITCH 'POD1','**.......*'"
```

Query Syntax: :MACHine1:TTRace:GLITch? <label_name>

Returned Format: [:MACHine1:TTRace:GLITch] <label_name>,<glitch_spec><NL>

Example:
```
10 DIM G$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:GLITCH? 'POD1'"
30 ENTER XXX;G$
40 PRINT G$
50 END
```

PATTern command/query

The PATTern command allows you to construct a pattern recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in the given pattern; therefore, a complete timing trace specification could require several commands. Since a label can contain up to 32 bits, the range of the pattern value will be between 0 and (2^{32})-1. The value may be expressed in binary (#B), octal (#Q), hexadecimal (#H) or decimal (default). When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. Since a pattern value can contain don't cares, the pattern specification parameter is handled as a string of characters instead of a number.

The PATTern query returns the pattern specification for the specified label in the base previously defined for the label.

Command Syntax: :MACHine{1|2}:TTRace:PATTern <label_name>,<pattern_spec>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<pattern_spec> ::= "{#B{0|1|X}...|
#Q{0|1|2|3|4|5|6|7|X}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
{0|1|2|3|4|5|6|7|8|9}...}"

Example:

```
OUTPUT XXX; ":MACHINE1:TTRACE:PATTERN 'DATA', '255'"
```

PATTern

Query Syntax: :MACHine{1|2}:TTRace:PATTern? <label_name>

Returned Format: [:MACHine{1|2}:TTRace:PATTern] <label_name>,<pattern_spec><NL>

Example:

```
10 DIM P$[100]
20 OUTPUT XXX; ":MACHINE2:TTRACE:PATTERN? 'DATA'"
30 ENTER XXX;P$
40 PRINT P$
50 END
```

TWAVeform Subsystem 19

Introduction

The TWAVeform subsystem contains the commands available for the Timing Waveforms menu in the HP 1652B/53B. These commands are:

- ACCumulate
- DELay
- INSert
- MMODe
- OCONdition
- OPATtern
- OSEarch
- OTIMe
- RANGe
- REMove
- RUNTil
- SPERiod
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XCONdition
- XOTime
- XPATtern
- XSEarch
- XTIMe

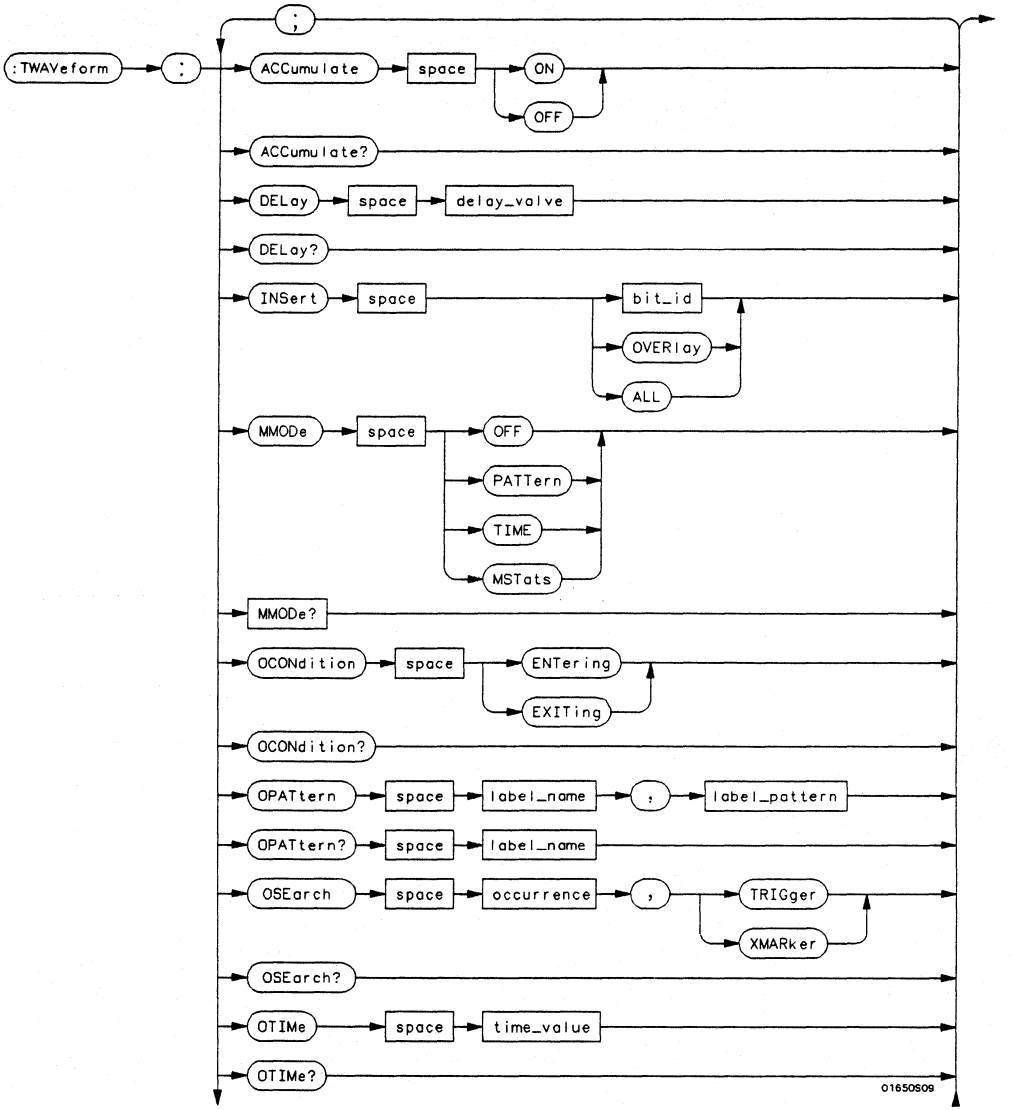


Figure 19-1. TWAVeform Subsystem Syntax Diagram

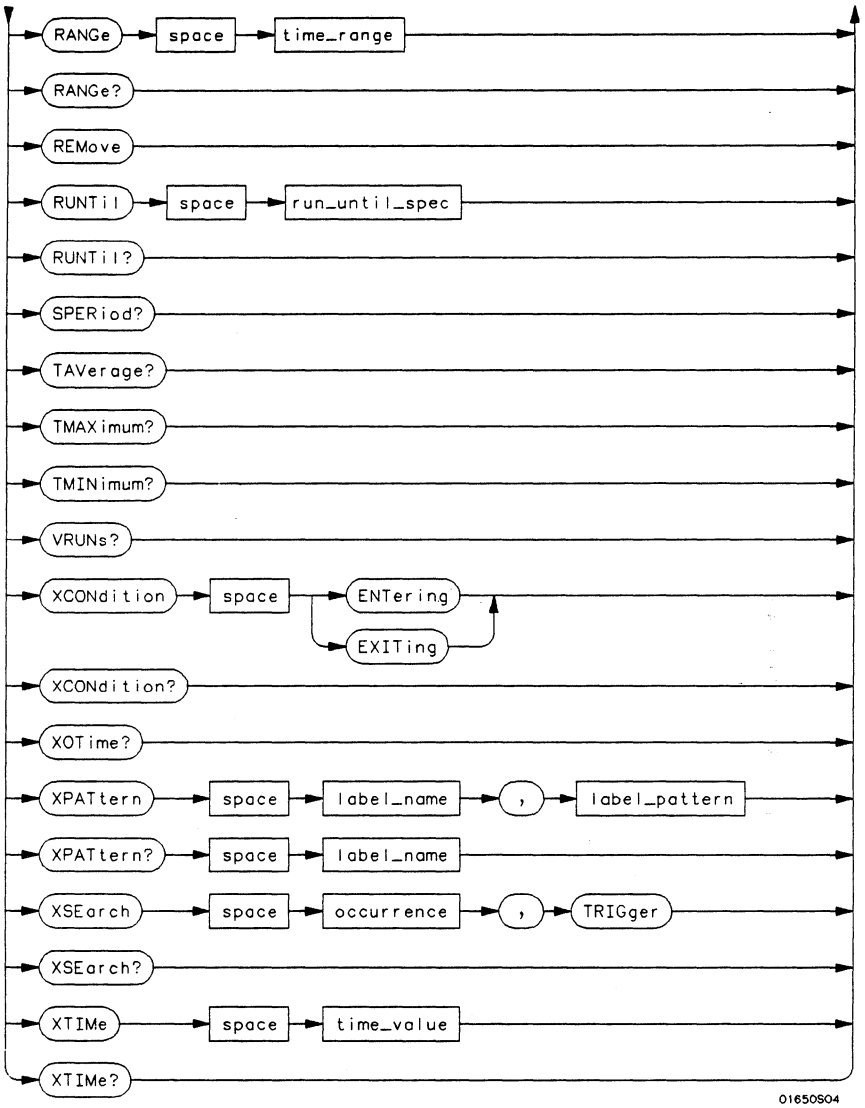


Figure 19-1. TWAVeform Subsystem Syntax Diagram (continued)

delay_value = *real number between -2500 s and +2500 s*
module_spec = *{1|2|3|4|5}*
bit_id = *integer from 0 to 31*
waveform = *string containing <acquisition_spec>{1|2}*
acquisition_spec = *{A|B|C|D|E} (slot where acquisition card is located)*
label_name = *string of up to 6 alphanumeric characters*
label_pattern = "*{#B{0|1|X}... |*
#Q{0|1|2|3|4|5|6|7|X}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}... |
{0|1|2|3|4|5|6|7|8|9}... }"
occurrence = *integer*
time_value = *real number*
label_id = *string of one alpha and one numeric character*
module_num = *slot number in which the timebase card is installed*
time_range = *real number between 100 ns and 10 ks*
run_until_spec = *{OFF|LT,<value>|GT,<value>|INRange<value>,<value>|*
OUTRange<value>,<value>}
GT = *greater than*
LT = *less than*
value = *real number*

Figure 19-1. TWAVeform Subsystem Syntax Diagram (continued)

TWAVeform Selector

The TWAVeform selector is used as part of a compound header to access the settings found in the Timing Waveforms menu. It always follows the MACHine selector because it selects a branch below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:TWAVeform

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY 100E-9"
```

ACCumulate command/query

The ACCumulate command allows you to control whether the chart display gets erased between each individual run or whether subsequent waveforms are allowed to be displayed over the previous ones.

The ACCumulate query returns the current setting. The query always shows the setting as the character "0" (off) or "1" (on).

Command Syntax: :MACHine{1|2}:TWAVeform:ACCumulate <setting>

where:

<setting> ::= {0|OFF} or {1|ON}

Example:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:ACCUMULATE ON"
```

Query Syntax: :MACHine{1|2}:TWAVeform:ACCumulate?

Returned Format: [:MACHine{1|2}:TWAVeform:ACCumulate] {0|1}<NL>

Example:
```
10 DIM P$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:ACCUMULATE?"
30 ENTER XXX; P$
40 PRINT P$
50 END
```

DELay

command/query

The DELay command specifies the amount of time between the timing trigger and the horizontal center of the the timing waveform display. The allowable values for delay are −2500 s to +2500 s. In glitch acquisition mode, as delay becomes large in an absolute sense, the sample rate is adjusted so that data will be acquired in the time window of interest. In transitional acquisition mode, data may not fall in the time window since the sample period is fixed at 10 ns and the amount of time covered in memory is dependent on how frequent the input signal transitions occur.

The DELay query returns the current time offset (delay) value from the trigger.

Command Syntax: :MACHine{1|2}:TWAVeform:DELay <delay_value>

where:

<delay_value> ::= real number between -2500 s and +2500 s

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY 100E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:DELay?

Returned Format: [:MACHine{1|2}:TWAVeform:DELay] <time_value><NL>

Example:

```
10 DIM D1$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY?"
30 ENTER XXX; D1$
40 PRINT D1$
50 END
```

INSert command

The INSert command inserts waveforms in the timing waveform display. The waveforms are added from top to bottom. When 24 waveforms are present, inserting additional waveforms replaces the last waveform .

The first parameter specifies the label name that will be inserted. The second parameter specifies the label bit number or overlay.

If OVERLAY is specified, all the bits of the label are displayed as a composite overlaid waveform.

Command Syntax: :MACHine{1|2}:TWAVeform:INSert <label_name> {<bit_id>|OVERlay}

where:

<label_name> ::= string of up to 6 alphanumeric characters
<bit_id> ::= integer from 0 to 31

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:INSERT 'WAVE',10"
```

MMODe command/query

The MMODe (Marker Mode) command selects the mode controlling marker movement and the display of the marker readouts. When PATTern is selected, the markers will be placed on patterns. When TIME is selected, the markers move on time. In MSTats, the markers are placed on patterns, but the readouts will be time statistics.

The MMODe query returns the current marker mode.

Command Syntax: :MACHine{1|2}:TWAVeform:MMODe {OFF|PATTern|TIME|MSTats}

Example:
```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:MMODE TIME"
```

Query Syntax: :MACHine{1|2}:TWAVeform:MMODe?

Returned Format: [:MACHine{1|2}:TWAVeform:MMODe] <marker_mode><NL>

where:

<marker_mode> ::= {OFF|PATTern|TIME|MSTats}

Example:
```
10 DIM M$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:MMODE?"
30 ENTER XXX; M$
40 PRINT M$
50 END
```

OCONdition command/query

The OCONdition command specifies where the O marker is placed. The O marker can be placed on the entry or exit point of the OPATtern when in the PATTern marker mode.

The OCONdition query returns the current setting.

Command Syntax: :MACHine{1|2}:TWAVeform:OCONdition {ENTering|EXITing}

Example:
```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OCONDITION ENTERING"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OCONdition?

Returned Format: [:MACHine{1|2}:TWAVeform:OCONdition] {ENTering|EXITing}<NL>

Example:
```
10 DIM Oc$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OCONDITION?"
30 ENTER XXX; Oc$
40 PRINT Oc$
50 END
```

OPATtern command/query

The OPATtern command allows you to construct a pattern recognizer term for the O marker which is then used with the OSEarch criteria and OCONdition when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and 2^{32} - 1, since a label may not have more than 32 bits. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

The OPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the O marker for a given label. If the O marker is not placed on valid data, don't cares (XX...X) are returned.

Command Syntax: :MACHine{1|2}:TWAVeform:OPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<label_pattern> ::= "{#B{0|1|X}...|
#Q{0|1|2|3|4|5|6|7|X}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
{0|1|2|3|4|5|6|7|8|9}...}"

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OPATTERN 'A','511'"
```

OPATtern

Query Syntax: :MACHine{1|2}:TWAVeform:OPATtern? <label_name>

Returned Format: [:MACHine{1|2}:TWAVeform:OPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Op$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OPATTERN? 'A'"
30 ENTER XXX; Op$
40 PRINT Op$
50 END
```

OSEarch

command/query

The OSEarch command defines the search criteria for the O marker which is then used with the associated OPATtern recognizer specification and the OCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search with the trigger or with the X marker. The actual occurrence the marker searches for is determined by the occurrence parameter of the OPATtern recognizer specification, relative to the origin. An occurrence of 0 places a marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

The OSEarch query returns the search criteria for the O marker.

Command Syntax: :MACHine{1|2}:TWAVeform:OSEarch <occurrence>,<origin>

where:

<origin> ::= {TRIGger|XMARker}
<occurrence> ::= integer from -9999 to +9999

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OSEARCH +10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OSEarch?

Returned Format: [:MACHine{1|2}:TWAVeform:OSEarch] <occurrence>,<origin><NL>

Example:

```
10 DIM Os$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OSEARCH?"
30 ENTER XXX; Os$
40 PRINT Os$
50 END
```

OTIMe command/query

The OTIMe command positions the O marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

The OTIMe query returns the O marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :MACHine{1|2}:TWAVeform:OTIMe <time_value>

where:

<time_value> ::= real number -2.5Ks to +2.5Ks

Example:
```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OTIME 30.0E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OTIMe?

Returned Format: [:MACHine{1|2}:TWAVeform:OTIMe] <time_value><NL>

Example:
```
10 DIM Ot$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OTIME?"
30 ENTER XXX; Ot$
40 PRINT Ot$
50 END
```

RANGe command/query

The RANGe command specifies the full-screen time in the timing waveform menu. It is equivalent to ten times the seconds-per-division setting on the display. The allowable values for RANGe are from 100 ns to 10 ks.

The RANGe query returns the current full-screen time.

Command Syntax: :MACHine{1|2}:TWAVeform:RANGe <time_value>

where:

<time_range> ::= real number between 100 ns and 10 ks

Example:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE 100E-9"
```

Query Syntax: :MACHine{1|2}:TWAVeform:RANGe?

Returned Format: [:MACHine{1|2}:TWAVeform:RANGe] <time_value><NL>

Example:
```
10 DIM Rg$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
30 ENTER XXX; Rg$
40 PRINT Rg$
50 END
```

REMove command

The REMove command deletes all waveforms from the display.

Command Syntax: :MACHine{1|2}:TWAVeform:REMove

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:REMOVE"
```

RUNTil command/query

The RUNTil (run until) command defines stop criteria based on the time between the X and O markers when the trace mode is in repetitive. When OFF is selected, the analyzer will run until either the "STOP" touch screen field is touched or the STOP command is sent. Run until the time between X and O marker options are:

- Less Than (LT) a specified time value
- Greater Than (GT) a specified time value
- In the range (INRange) between two time values
- Out of the range (OUTRange) between two time values

End points for the INRange and OUTRange should be at least 10 ns apart since this is the minimum time at which data is sampled.

This command affects the timing analyzer only, and has no relation to the RUNTil commands in the SLISt and COMPare subsystems.

The RUNTil query returns the current stop criteria.

Command Syntax: :MACHine{1|2}:TWAVeform:RUNTil <run_until_spec>

where:

<run_until_spec> ::= {OFF | LT,<value> | GT,<value> | INRange<value>,<value> | OUTRange<value>,<value>}

<value> ::= real number

Examples:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL GT, 800.0E-6"
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL INRANGE, 4.5, 5.5"
```

RUNTil

Query Syntax: :MACHine{1|2}:TWAVeform:RUNTil?

Returned Format: [:MACHine{1|2}:TWAVeform:RUNTil] <run_until_spec><NL>

Example:

```
10 DIM Ru$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL?"
30 ENTER XXX; Ru$
40 PRINT Ru$
50 END
```

SPERiod

query

The SPERiod query returns the sample period of the last run.

Query Syntax: :MACHine{1|2}:TWAVeform:SPERiod?

Returned Format: [:MACHine{1|2}:TWAVeform:SPERiod] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Sp$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:SPERIOD?"
30 ENTER XXX; Sp$
40 PRINT Sp$
50 END
```

TAVerage query

The TAVerage query returns the value of the average time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:TAVerage?

Returned Format: [:MACHine{1|2}:TWAVeform:TAVerage] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tv$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:TAVERAGE?"
30 ENTER XXX; Tv$
40 PRINT Tv$
50 END
```

TMAXimum query

The TMAXimum query returns the value of the maximum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:TMAXimum?

Returned Format: [:MACHine{1|2}:TWAVeform:TMAXimum] <time_value><NL>

where

<time_value> ::= real number

Example:

```
10 DIM Tx$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:TMAXIMUM?"
30 ENTER XXX; Tx$
40 PRINT Tx$
50 END
```

TMINimum query

The TMINimum query returns the value of the minimum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:TMINimum?

Returned Format: [:MACHine{1|2}:TWAVeform:TMINimum] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tm$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:TMINIMUM?"
30 ENTER XXX; Tm$
40 PRINT Tm$
50 END
```

VRUNs query

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Query Syntax: :MACHine{1|2}:TWAVeform:VRUNs?

Returned Format: [:MACHine{1|2}:TWAVeform:VRUNs] <valid_runs>,<total_runs><NL>

where:

<valid_runs> ::= zero or positive integer
<total_runs> ::= zero or positive integer

Example:

```
10 DIM Vr$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:VRUNS?"
30 ENTER XXX; Vr$
40 PRINT Vr$
50 END
```

XCONdition command/query

The XCONdition command specifies where the X marker is placed. The X marker can be placed on the entry or exit point of the XPATtern when in the PATTern marker mode.

The XCONdition query returns the current setting.

Command Syntax: :MACHine{1|2}:TWAVeform:XCONdition {ENTering|EXITing}

Example:
```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XCONDITION ENTERING"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XCONdition?

Returned Format: [:MACHine{1|2}:TWAVeform:XCONdition] {ENTering|EXITing}<NL>

Example:
```
10 DIM Xc$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XCONDITION?"
30 ENTER XXX; Xc$
40 PRINT Xc$
50 END
```

XOTime query

The XOTime query returns the time from the X marker to the O marker. If data is not valid, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:XOTime?

Returned Format: [:MACHine{1|2}:TWAVeform:XOTime] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Xot$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XOTIME?"
30 ENTER XXX; Xot$
40 PRINT Xot$
50 END
```

XPATtern command/query

The XPATtern command allows you to construct a pattern recognizer term for the X marker which is then used with the XSEarch criteria and XCONdition when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and 2^{32} - 1, since a label may not have more than 32 bits. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

The XPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the X marker for a given label. If the X marker is not placed on valid data, don't cares (XX...X) are returned.

Command Syntax: :MACHine{1|2}:TWAVeform:XPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<label_pattern> ::= "{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XPATTERN 'A','511'"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XPATtern? <label_name>

Returned Format: [:MACHine{1|2}:TWAVeform:XPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Xp$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XPATTERN? 'A'"
30 ENTER XXX; Xp$
40 PRINT Xp$
50 END
```

XSEarch command/query

The XSEarch command defines the search criteria for the X marker which is then used with the associated XPATtern recognizer specification and the XCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search with the trigger. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 (zero) places a marker on the origin.

The XSEarch query returns the search criteria for the X marker.

Command Syntax: :MACHine{1|2}:TWAVeform:XSEarch <occurrence>,<origin>

where:

<origin> ::= TRIGger
<occurrence> ::= integer from -9999 to +9999

Example:
```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XSEARCH,+10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XSEarch? <occurrence>,<origin>

Returned Format: [:MACHine{1|2}:TWAVeform:XSEarch] <occurrence>,<origin><NL>

Example:
```
10 DIM Xs$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XSEARCH?"
30 ENTER XXX; Xs$
40 PRINT Xs$
50 END
```

XTIMe

command/query

The XTIMe command positions the X marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

The XTIMe query returns the X marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :MACHine{1|2}:TWAVeform:XTIMe <time_value>

where:

<time_value> ::= real number from −2.5Ks to +2.5Ks

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XTIME 40.0E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XTIMe?

Returned Format: [:MACHine{1|2}:TWAVeform:XTIMe] <time_value><NL>

Example:

```
10 DIM Xt$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XTIME?"
30 ENTER XXX; Xt$
40 PRINT Xt$
50 END
```

SYMBol Subsystem 20

Introduction

The SYMBol subsystem contains the commands that allow you to define symbols on the controller and download them to the HP 1652B/53B logic analyzer. The commands in this subsystem are:

- BASE
- PATTern
- RANGe
- REMove
- WIDTh

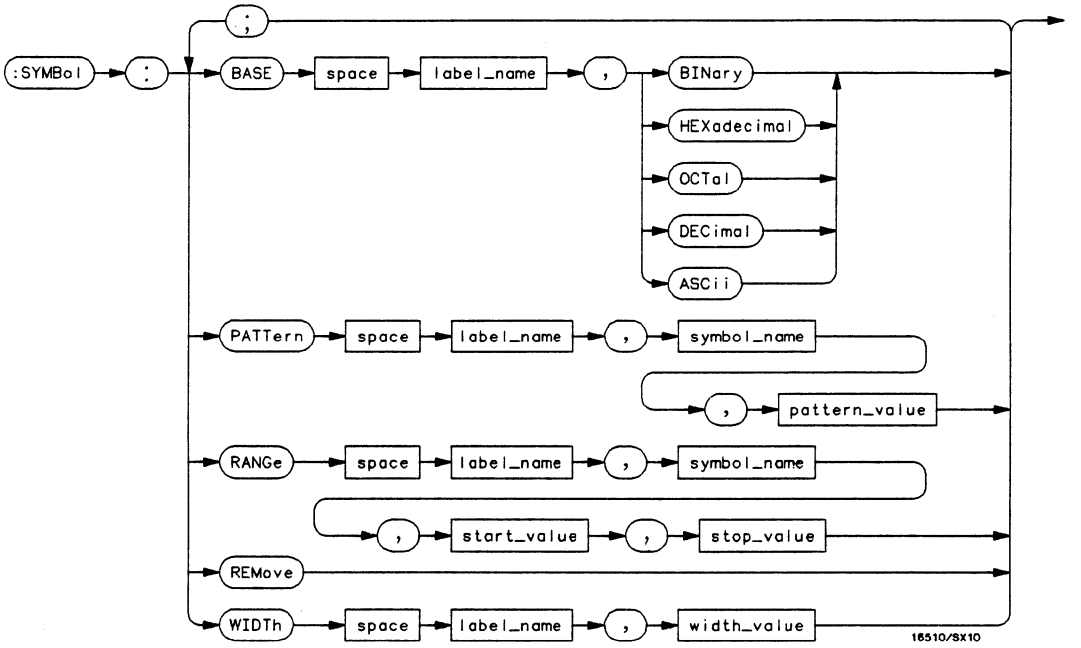


Figure 20-1. SYMBol Subsystem Diagram

label_name = *string of up to 6 alphanumeric characters*
symbol_name = *string of up to 16 alphanumeric characters*
pattern_value = "{#B{0|1|X}... |
#Q{0|1|2|3|4|5|6|7|X}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}... |
{0|1|2|3|4|5|6|7|8|9}... }"
start_value = "{#B{0|1}... |
#Q{0|1|2|3|4|5|6|7}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}... |
{0|1|2|3|4|5|6|7|8|9}... }"
stop_value = "{#B{0|1}... |
#Q{0|1|2|3|4|5|6|7}... |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}... |
{0|1|2|3|4|5|6|7|8|9}... }"
width_value = *integer from 1 to 16*

Figure 20-1. SYMBol Subsystem Syntax Diagram (continued)

SYMBol selector

The SYMBol selector is used as a part of a compound header to access the commands used to create symbols. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SYMBol

Example:

```
OUTPUT XXX;":MACHINE1:SYMBOL:BASE 'DATA', BINARY"
```

BASE command

The BASE command sets the base in which symbols for the specified label will be displayed in the symbol menu. It also specifies the base in which the symbol offsets are displayed when symbols are used.

Note BINary is not available for labels with more than 20 bits assigned. In this case the base will default to HEXadecimal.

Command Syntax: :MACHine{1|2}:SYMBol:BASE <label_name>,<base_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<base_value> ::= {BINary | HEXadecimal | OCTal | DECimal | ASCii}

Example:

```
OUTPUT XXX;":MACHINE1:SYMBOL:BASE 'DATA',HEXADECIMAL"
```

PATTern command

The PATTern command allows you to create a pattern symbol for the specified label.

Because don't cares (X) are allowed in the pattern value, it must always be expressed as a string. You may still use different bases, though don't cares cannot be used in a decimal number.

Command Syntax: :MACHine{1|2}:SYMBol:PATTern<label_name>,<symbol_name>,<pattern_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<symbol_name> ::= string of up to 16 alphanumeric characters

<pattern_value> ::= "{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"

Example:

```
OUTPUT XXX;":MACHINE1:SYMBOL:PATTERN 'STAT', 'MEM_RD','#H01XX'"
```

RANGe command

The RANGe command allows you to create a range symbol containing a start value and a stop value for the specified label. The values may be in binary (#B), octal (#Q), hexadecimal (#H) or decimal (default). You may not use "don't cares" in any base.

Command Syntax: :MACHine{1|2}:SYMBol:RANGe <label_name>,<symbol_name>,<start_value>, <stop_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<symbol_name> ::= string of up to 16 alphanumeric characters

<start_value> ::= "{#B{0|1}...|
#Q{0|1|2|3|4|5|6|7}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...|
{0|1|2|3|4|5|6|7|8|9}...}"

<stop_value> ::= "{#B{0|1}...|
#Q{0|1|2|3|4|5|6|7}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...|
{0|1|2|3|4|5|6|7|8|9}...}"

Example:

```
OUTPUT XXX;":MACHINE1:SYMBOL:RANGE 'STAT', 'IO_ACC','0','#H000F'"
```

REMove command

The REMove command deletes all symbols from a specified machine.

Command Syntax: :MACHine{1|2}:SYMBol:REMove

Example:

```
OUTPUT XXX;":MACHINE1:SYMBOL:REMOVE"
```

WIDTh command

The WIDTh command specifies the width (number of characters) in which the symbol names will be displayed when symbols are used.

Note The WIDTh command does not affect the displayed length of the symbol offset value.

Command Syntax: :MACHine{1|2}:SYMBol:WIDTh <label_name>,<width_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<width_value> ::= integer from 1 to 16

Example:

```
OUTPUT XXX;":MACHINE1:SYMBOL:WIDTH 'DATA',9 "
```

SCOPe Subsystem 21

Introduction

The SCOPe subsystem provides access to the commands and the oscilloscope subsystem commands that control the basic operation of the oscilloscope. At the SCOPe subsystem level is a command that turns the oscilloscope on or off (SMODe), specifies how the oscilloscope is Armed (ARM), and the AUToscale command.

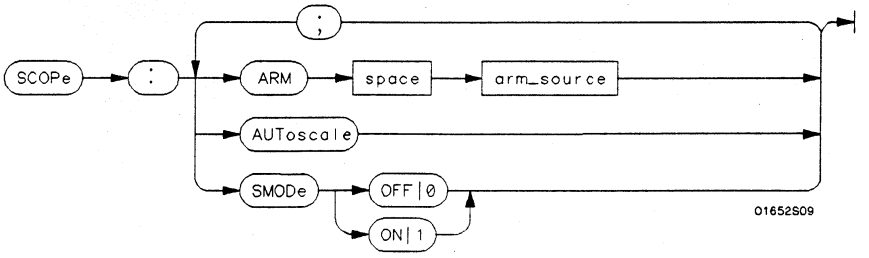
Additionally, the following subsystems are a part of the SCOPe subsystem. Each is explained in a separate chapter.

- CHANnel subsystem (chapter 22)
- TRIGger subsystem (chapter 23)
- ACQuire subsystem (chapter 24)
- TIMebase subsystem (chapter 25)
- WAVeform subsystem (chapter 26)
- MEASure subsystem (chapter 27)

Not all scope-related functions can be duplicated with programming instructions. If you are unable to get a desired configuration strictly through programming instruction, try the following steps:

1. Manually configure the HP 1652B/53B through the front panel.
2. Save configuration to a disk (through the front panel or through the `:MMEM:STORE "CONFIG","Setups"` instruction).

Now you can use the command `MMEM:LOAD "CONFIG"` to load in the desired configuration.



arm_source = {*RUN* | *MACHine*{*1* | *2*} | *BNC*}

Figure 21-1. SCOPe Subsystem Syntax Diagram

SCOPe selector

The SCOPe selector is used to indicate the beginning of a compound command (or query) for a function within the SCOPe subsystem. Since SCOPe is a root-level command, it will normally appear as the first element of a compound header.

Command Syntax: :SCOPe

Example:

```
OUTPUT XXX; ":SCOPE:TRIGGER:SLOPE NEGATIVE"
```

Arm command/query

The ARM command specifies the arming source of the oscilloscope.

The ARM query returns the source that the oscilloscope is armed by.

Command Syntax: :SCOPe:ARM <arm_source>

where:

<arm_source> ::= {RUN | MACHine{1|2} | BNC}

Example:
```
OUTPUT XXX;":SCOPE:ARM:MACHINE2"
```

Query Syntax: :SCOPe:ARM?

Returned Format: [:SCOPe:ARM] <arm_source>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":SCOPE:ARM?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

AUToscale command

The AUToscale command causes the oscilloscope to automatically select the vertical sensitivity, vertical offset, trigger level and timebase settings for a stable display on one or both channels. The input signal required for Autoscale must have an amplitude above 10 mV peak, and a frequency between 50 Hz and 100 MHz..

Command Syntax: :SCOPe:AUToscale

Example:

```
OUTPUT XXX;":SCOPE:AUTOSCALE"
```

SMODe command/query

The SMODe command allows the oscilloscope to be turned on or off over the bus.

The SMODe query returns the current status of the oscillosocpe.

Command Syntax: :SCOPe:SMODe {ON|OFF}

Example:
```
OUTPUT XXX;":SCOPe:SMODe ON"
```

Query Syntax: :SCOPe:SMODe?

Returned Format: [:SCOPe:SMODe] {ON|OFF}<NL>

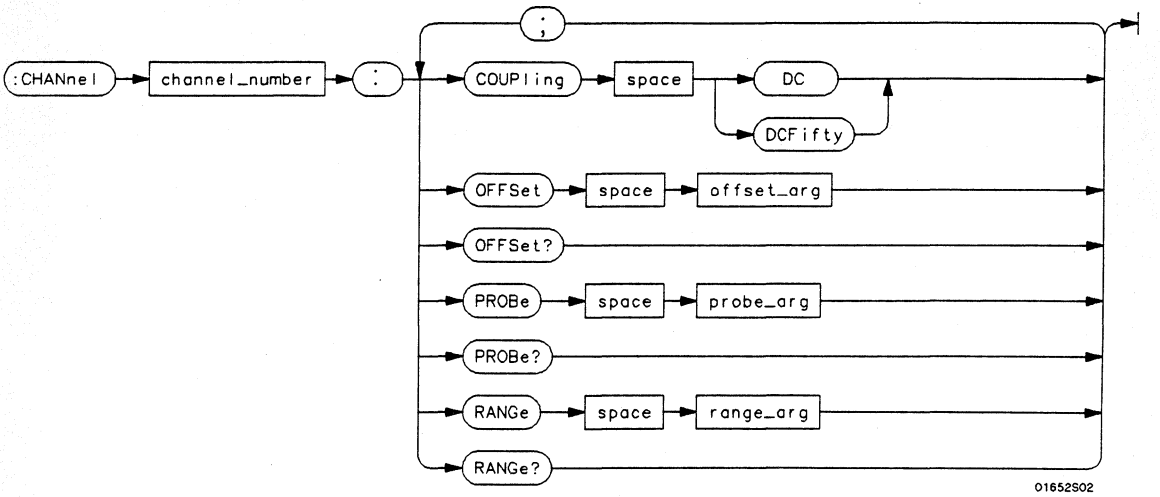
Example:
```
10 DIM Sm$[100]
20 OUTPUT XXX;":SCOPE:SMODE?"
30 ENTER XXX;Sm$
40 PRINT Sm$
50 END
```

CHANnel Subsystem 22

Introduction

The CHANnel subsystem commands control the channel display and the vertical axis of the oscilloscope. Each channel must be programmed independently for all offset, range and probe functions. The commands are:

- CHANnel
- COUPling
- OFFSet
- PROBe
- RANGe



channel_number = {*1* | *2*}
offset_arg = *real number defining the voltage at the center of the display. The offset range depends on the input impedance setting. The offset range for 1 MΩ input is −125 V to +125 V. The offset range for 50 Ω input is −5 V to + 5 V.*
probe_arg = *integer from 1 through 1000, specifying the probe attenuation with respect to 1.*
range_arg = *real number specifying vertical sensitivity. The allowable range is 15 mV to 10 V for a probe attenuation of 1. The specified range is equal to 4 times Volts/Div.*

Figure 22-1. CHANnel Subsystem Syntax Diagram

CHANnel selector

The CHANnel selector is used as part of a compound command header to access the settings found in oscilloscope's CHANnel menu. It always follows the SCOPe selector because it selects a branch below the SCOPe level in the command tree.

Command Syntax: :SCOPe:CHANnel<N>

where:

<N> ::= {1 | 2}

Example:

```
OUTPUT XXX; ":SCOPE:CHANNEL2:OFFSET 2.5"
```

COUPling command/query

The COUPling command sets the input impedance for the selected channel. The choices are either 1M Ohm (DC) or 50 Ohms (DCFifty).

The query returns the current input impedance for the specified channel.

Command Syntax: :SCOPe:CHANnel{1|2}:COUPling {DC|DCFifty}

Example:
```
OUTPUT XXX;":SCOPE:CHANNEL1:COUPLING DC"
```

Query Syntax: :SCOPe:CHANnel{1|2}:COUPling?

Returned Format: [:SCOPe:CHANnel{1|2}:COUPling] {DC|DCFifty}<NL>

Example:
```
10 DIM Cc$[100]
20 OUTPUT XXX;":SCOPE:CHANNEL1:COUPLING?"
30 ENTER XXX;Cc$
40 PRINT Cc$
50 END
```

OFFSet

command/query

The OFFSet command sets the voltage that is represented at center screen for the selected channel. The allowable offsets for 1:1 probes are:

± 2 V < 74 mV/div
± 10 V between 74 mV/div and 370 mV/div
± 50V between 370 mV/div and 1.85 V/div
± 125 V > 1.85 V/div

When the input impedance is set to 50 Ω the maximum offset is æ 2 V for V/Div settings less than 74 mV and is ±5 V for V/Div settings greater than 74 mV.

The offset value is recompensated whenever the probe attenuation factor is changed.

The query returns the current value for the selected channel.

Command Syntax: :SCOPe:CHANnel{1|2}:OFFSet <value>

where:

<value> ::= {− 250V to + 250 V max. at 1 MΩ | − 5 V to + 5 V at 50 Ω}

Example:
```
OUTPUT XXX;":SCOP:CHAN1:OFFS 1.5"
```

Query Syntax: :SCOPe:CHANnel{1|2}:OFFSet?

Returned Format: [:SCOPe:CHANnel{1|2}:OFFSet] <value><NL>

Example:
```
10 DIM Co$[100]
20 OUTPUT XXX;":SCOPE:CHANNEL1:OFFSET?"
30 ENTER XXX;Co$
40 PRINT Co$
50 END
```

PROBe command/query

The PROBe command specifies the attenuation factor for an external probe connected to a channel. The command changes the channel voltage references such as range, offset, trigger levels and automatic measurements. The actual sensitivity is not changed at the channel input. The allowable probe attenuation factor is an integer from 1 to 1000.

The query returns the probe attenuation factor for the selected channel.

Command Syntax: :SCOPe:CHANnel{1|2}:PROBe <atten>

where:

<atten> ::= integer from 1 to 1000

Example:
```
OUTPUT XXX;":SCOPe:CHAN1:PROB 10"
```

Query Syntax: :SCOPe:CHANnel{1|2}:PROBe?

Returned Format: [:SCOPe:CHANnel{1|2}:PROBe] <atten><NL>

Example:
```
10 DIM Att$[100]
20 OUTPUT XXX;":SCOPE:CHANNEL1:PROBE?"
30 ENTER XXX;Att$
40 PRINT Att$
50 END
```

RANGe

command/query

The RANGe command defines the full-scale (4 × Volts/Div) vertical axis of the selected channel. The values for the RANGe command are dependent on the current probe attenuation factor for the selected channel. The allowable range for a probe attenuation factor of 1:1 is 60 mV to 40 V. For a larger probe attenuation factor, multiply the range limit by the probe attenuation factor.

The RANGe query returns the current range setting.

Command Syntax: :SCOPe:CHANnel{1|2}:RANGe <range>

where:

<range> ::= 60 mV to 40 V for a probe attenuation factor of 1:1

Example:

```
OUTPUT XXX;":SCOPE:CHANNEL1:RANGE 4.8"
```

Query Syntax: :SCOPe:CHANnel{1|2}:RANGe?

Returned Format: [:SCOPe:CHANnel{1|2}:RANGe] <range><NL>

Example:

```
10 DIM Pr$[100]
20 OUTPUT XXX;":SCOPE:CHANNEL1:RANGE?"
30 ENTER XXX;Pr$
40 PRINT Pr$
50 END
```

TRIGger Subsystem 23

Introduction

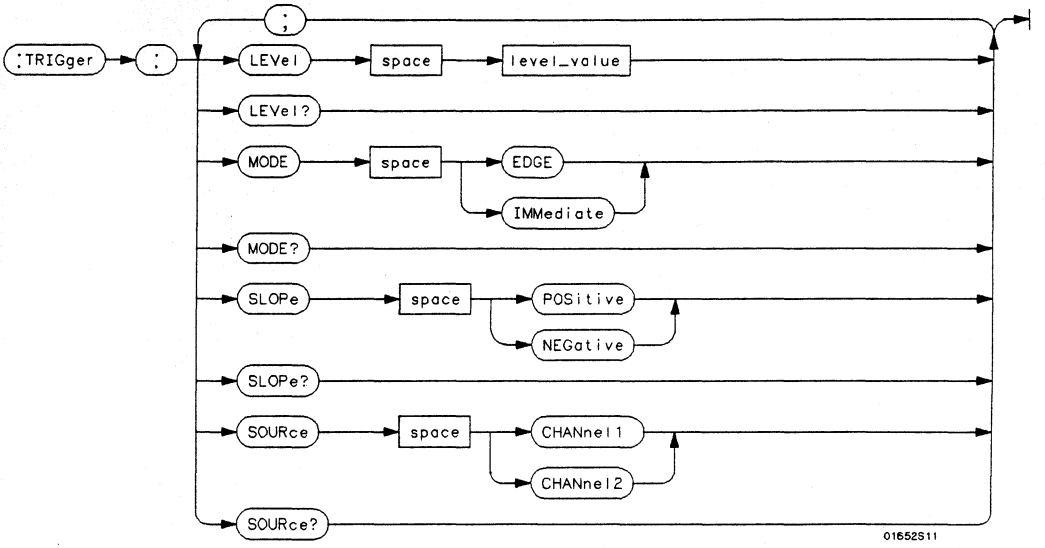
The commands of the TRIGger subsystem allow you to set all the trigger conditions necessary for generating a trigger. There are two trigger modes: Edge and Immediate. If a command is valid for the chosen trigger mode, then that setting will be accepted by the oscilloscope. However, if the command is not valid for the trigger mode, an error will be generated. None of the commands of this subsystem are used in conjunction with Immediate trigger mode. See Figure 23-1 for the TRIGger subsystem syntax diagram.

The Edge Trigger Mode

In the Edge trigger mode, the oscilloscope triggers on an edge of a waveform, specified by the SOURce, LEVel, and SLOPe commands. If a source is not specified, then the current source is assumed.

The Immediate Trigger Mode

In the Immediate trigger mode, the oscilloscope will trigger by itself when the arming requirements are met.



level _value = *trigger level in volts*

Figure 23-1. TRIGger Subsystem Syntax Diagram

TRIGger selector

The TRIGger selector is used as part of a compound command header to access the settings found in oscilloscope's Trigger menu. It always follows the SCOPe selector because it selects a branch below the SCOPe level in the command tree.

Command Syntax: :SCOPe:TRIGger

Example:

```
OUTPUT XXX; ":SCOPE:TRIGGER:CHANNEL1;LEVEL 2.0"
```

LEVEL command/query

The LEVEL command sets the trigger level voltage for the selected source or path. This command cannot be used in the IMMEDIATE trigger mode.

The query returns the trigger level for the current path or source.

Note There is no shortform for LEVEL. This is an intentional deviation from the normal truncation rule.

Command Syntax: :SCOPe:TRIGger:LEVEL <value>

where:

<value> ::= Trigger level in volts

Example:
```
OUTPUT XXX;":SCOPE:TRIG:LEVEL 1.0"
```

Query Syntax: :SCOPe:TRIGger:LEVEL?

Returned Format: [:SCOPe:TRIGger:LEVEL] <value><NL>

Example:
```
10 DIM E1$[100]
20 OUTPUT XXX;":SCOPE:TRIGGER:SOURCE CHANNEL1;LEVEL?"
30 ENTER XXX;E1$
40 PRINT E1$
50 END
```

MODE

command/query

The MODE command allows you to select the trigger mode for the oscilloscope. The EDGE mode will trigger the oscilloscope on an edge whose slope is determined by the SLOPe command at a voltage set by the LEVEL command. In the IMMediate trigger mode, the oscilloscope goes to a freerun mode and does not wait for a trigger. The IMMediate mode is used in armed-by other machine applications.

The query returns the current mode.

Command Syntax: :SCOPe:TRIGger:MODE {EDGE|IMMediate}

Example:
```
OUTPUT XXX;":SCOPE:TRIGGER:MODE EDGE"
```

Query Syntax: :SCOPe:TRIGger:MODE?

Returned Format: [:SCOPe:TRIGger:MODE] {EDGE|IMMediate}<NL>

Example:
```
10 DIM Md$[100]
20 OUTPUT XXX;":SCOPE:TRIGGER:MODE?"
30 ENTER XXX;Md$
40 PRINT Md$
50 END
```

SLOPe command/query

The SLOPe command selects the trigger slope for the previously specified trigger source. This command can only be used in the EDGE trigger mode.

The query returns the slope of the current trigger source.

Command Syntax: :SCOPe:TRIGger:SLOPe {POSitive|NEGative}

Example:
```
OUTPUT XXX;":SCOP:TRIG:SOURCE CHAN1;SLOPE POS"
```

Query Syntax: :SCOPe:TRIGger:SLOPe?

Returned Format: [:SCOPe:TRIGger:SLOPe] {POSitive|NEGative}<NL>

Example:
```
10 DIM Ts$[100]
20 OUTPUT XXX;":SCOP:TRIG:SOUR CHAN1;SLOP?"
30 ENTER XXX;Ts$
40 PRINT Ts$
50 END
```

SOURce command/query

The SOURce command is used to select the trigger source and is used for any subsequent SLOPe and LEVEL commands. This command can only be used in the EDGE trigger mode.

The query returns the current trigger source.

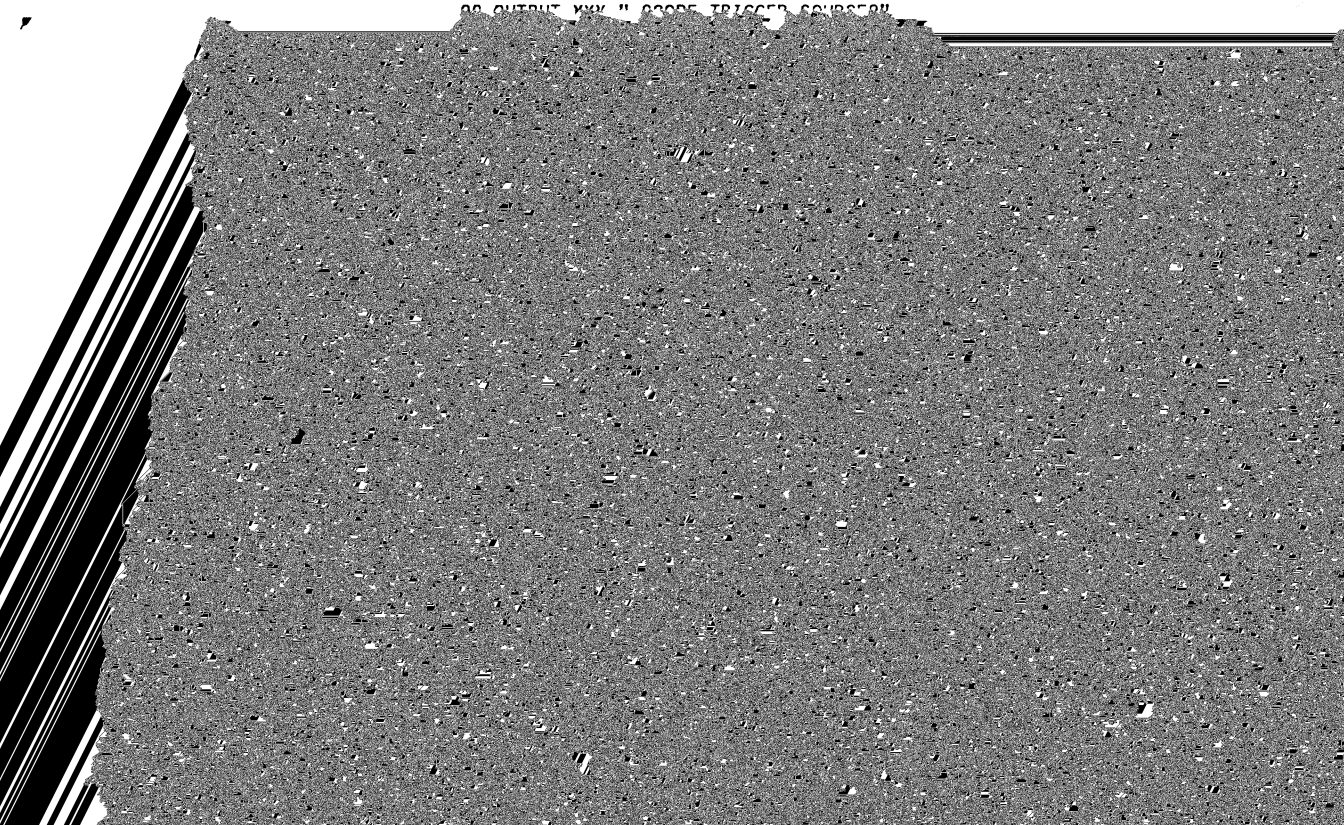
Command Syntax: :SCOPe:TRIGger:SOURce {CHANnel{1|2}}

Example:
```
OUTPUT XXX;":SCOP:TRIG:SOUR CHAN1"
```

Query Syntax: :SCOPe:TRIGger:SOURce?

Returned Format: [:SCOPe:TRIGger:SOURce] {CHANnel{1|2}}<NL>

Example:
```
10 DIM Tso$[100]
```

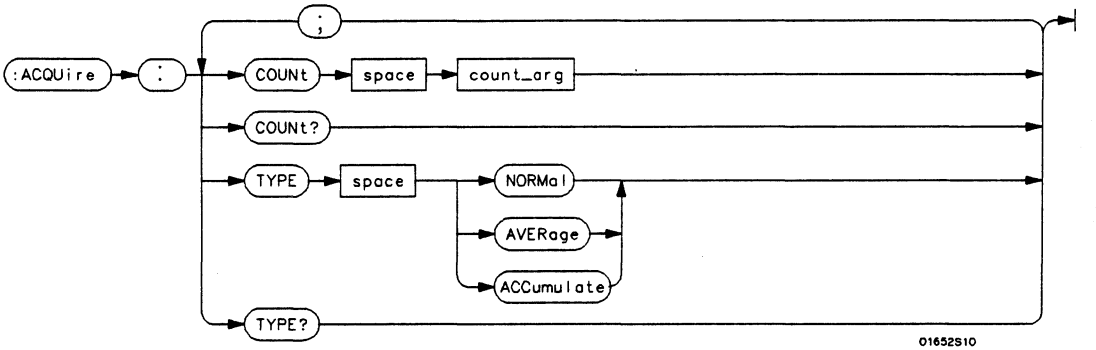


ACQuire Subsystem 24

Introduction

The ACQuire subsystem commands are used to select the type of acquisition and the number of averages to be taken if the average type is chosen. The commands are:

- COUNt
- TYPE



count_arg = {*2|4|8|16|32|64|128|256*} *An integer that specifies the number of averages to be taken of each time point.*

Figure 24-1. ACQuire Subsystem Syntax Diagram

Acquisition Type Normal

In the Normal mode, with the ACCumulate command OFF, the oscilloscope acquires waveform data and then displays the waveform. When the oscilloscope makes a new acquisition, the previously acquired waveform is erased from the display and replaced by the newly acquired waveform.

When the ACCumulate command is ON, the oscilloscope displays all the waveform acquisitions without erasing the previously acquired waveform.

Acquisition Type Average

In the Average mode, the oscilloscope averages the data points on the waveform with previously acquired data. Averaging helps eliminate random noise from the displayed waveform. In this mode the ACCumulate command is OFF. When Average mode is selected, the number of averages must also be specified using the COUNt command. Previously averaged waveform data is erased from the display and the newly averaged waveform is displayed.

ACQuire selector

The ACQuire selector is used as part of a compound command header to access the settings found in oscilloscope's Acquire menu. It always follows the SCOPe selector because it selects a branch below the SCOPe level in the command tree.

Command Syntax: :SCOPe:ACQuire

Example:

```
OUTPUT XXX; ":SCOPE:ACQUIRE:TYPE NORMAL"
```

COUNt command/query

The COUNt command specifies the number of acquisitions for the running weighted average. This command generates an error if Normal acquisition mode is specified.

The query returns the last specified count.

Command Syntax: :SCOPe:ACQuire:COUNt <count>

where

<count> ::= {2|4|8|16|32|64|128|256}

Example
```
OUTPUT XXX;":SCOPE:ACQUIRE:COUNT 16"
```

Query Syntax: :SCOPe:ACQuire:COUNt?

Returned Format [:SCOPe:ACQuire:COUNt] <count><NL>

Example:
```
10 DIM Ac$[100]
20 OUTPUT XXX;":SCOPE:ACQ:COUN?"
30 ENTER XXX;Ac$
40 PRINT Ac$
50 END
```

TYPE command/query

The TYPE command selects the type of acquisition that is to take place when the STARt command is executed. One of three acquisition types may be selected: the NORMal, AVERage, or ACCumulate mode.

The query returns the last specified type.

Command Syntax :SCOPe:ACQuire:TYPE {NORMal|AVERage|ACCumulate}

Example:
```
OUTPUT XXX;":SCOPE:ACQUIRE:TYPE NORMAL"
```

Query Syntax: :SCOPe:ACQuire:TYPE?

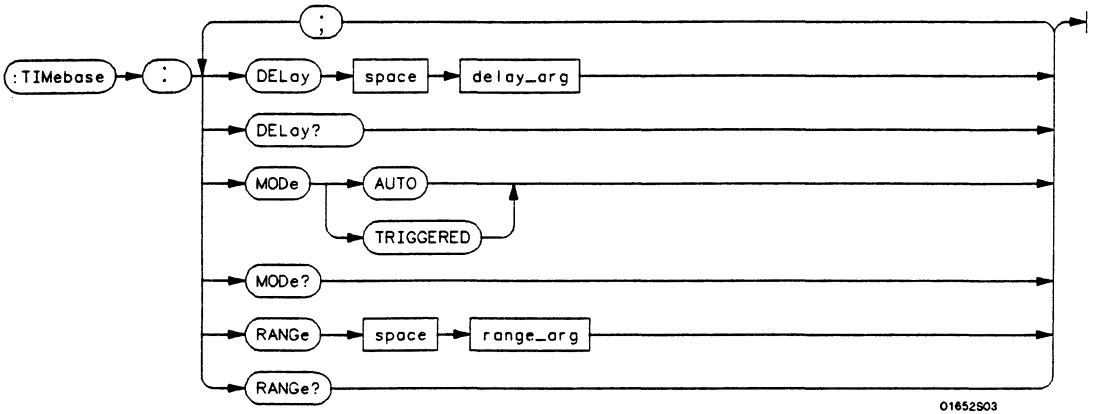
Returned Format: [:SCOPe:ACQuire:TYPE] {NORMal|AVERage}<NL>

Example:
```
10 DIM At$[100]
20 OUTPUT XXX;":SCOPE:ACQUIRE:TYPE?"
30 ENTER XXX;At$
40 PRINT At$
50 END
```

TIMebase Subsystem 25

Introduction

The commands of the TIMebase subsystem control the Timebase, Trigger Delay Time, and the Timebase Mode. If TRIGGERED mode is to be used, ensure that the trigger specifications of the TRIGger subsystem have been set. Refer to Figure 25-1 for the TIMebase subsystem syntax diagram.



delay_arg = *delay time in seconds, from -2500 seconds through +2500 seconds*
range_arg = *a real number from 5 ns through 10s*

Figure 25-1. TIMebase Subsystem Syntax Diagram

TIMebase selector

The TIMebase selector is used as part of a compound command header to access the settings found in oscilloscope's Timebase menu. It always follows the SCOPe selector because it selects a branch below the SCOPe level in the command tree.

Command Syntax: :SCOPe:TIMebase

Example:

```
OUTPUT XXX; ":SCOPE:TIMEBASE:MODE AUTO"
```

DELAY

command/query

The DELAY command sets the time between the trigger and the center of the screen if the trigger events count is zero. If the trigger events count is non-zero, the center of the screen is the trigger events count plus the delay time.

The query returns the current delay setting.

Note The DELAY command in the TIMebase subsystem has no shortform. This is an intentional deviation from the normal truncation rules.

Command Syntax: :SCOPe:TIMebase:DELAY <delay time>

where:

<delay time> ::= delay time in seconds

Example:
```
OUTPUT XXX;":SCOPe:TIMebase:DELAY 2US"
```

Query Syntax: :SCOPe:TIMebase:DELAY?

Returned Format: [:SCOPe:TIMebase:DELAY] <value><NL>

Example:
```
10 DIM Dt$[100]
20 OUTPUT XXX;":SCOPe:TIMebase:DELAY?"
30 ENTER XXX;Dt$
40 PRINT Dt$
50 END
```

MODE

command/query

The MODE command sets the oscilloscope timebase to either Auto or Triggered mode. When the AUTO mode is chosen, the oscilloscope waits approximately one second for a trigger to occur. If a trigger is not generated within that time, then auto trigger is executed. If a signal is not applied to the input, a baseline is displayed. If there is a signal at the input and the specified trigger conditions have not been met within one second, the waveform display will not be synchronized to a trigger.

When the TRIGGERED mode is chosen, the oscilloscope waits until a trigger is received before data is acquired. The TRIGGERED mode should be used when the trigger source signal is less than at a 40 Hz repetition rate.

The Auto-Trig On field in the trigger menu is the same as the AUTO mode over HP-IB or RS-232C. Setting the mode to TRIGGERED is the same as the Auto-Trig Off on the front panel.

The query returns the current TIMebase mode.

Note 

The TRIGGERED argument for MODE has no shortform. This is an intentional deviation from the normal truncation rule.

Command Syntax: :SCOPe:TIMebase:MODE {TRIGGERED|AUTO}

Example: OUTPUT XXX;":SCOPE:TIME:MODE AUTO"

Query Syntax: :SCOPe:TIMebase:MODE?

Returned Format: [:SCOPe:TIMebase:MODE] {AUTO|TRIGGERED}<NL>

Example:

```
10 DIM Tm$[100]
20 OUTPUT XXX;":SCOPe:TIMEBASE:MODE?"
30 ENTER XXX;Tm$
40 PRINT Tm$
50 END
```

RANGe command/query

The RANGE command sets the full-scale horizontal time in seconds. The RANGE value is ten times the front panel field of s/div.

The query returns the current range.

Command syntax: :SCOPe:TIMebase:RANGe <range>

where:

<range> ::= time in seconds

Example:
```
OUTPUT XXX;":SCOPE:TIMEBASE:RANGE 2US"
```

Query Syntax: :SCOPe:TIMebase:RANGe?

Returned Format: [:SCOPe:TIMebase:RANGe] <range><NL>

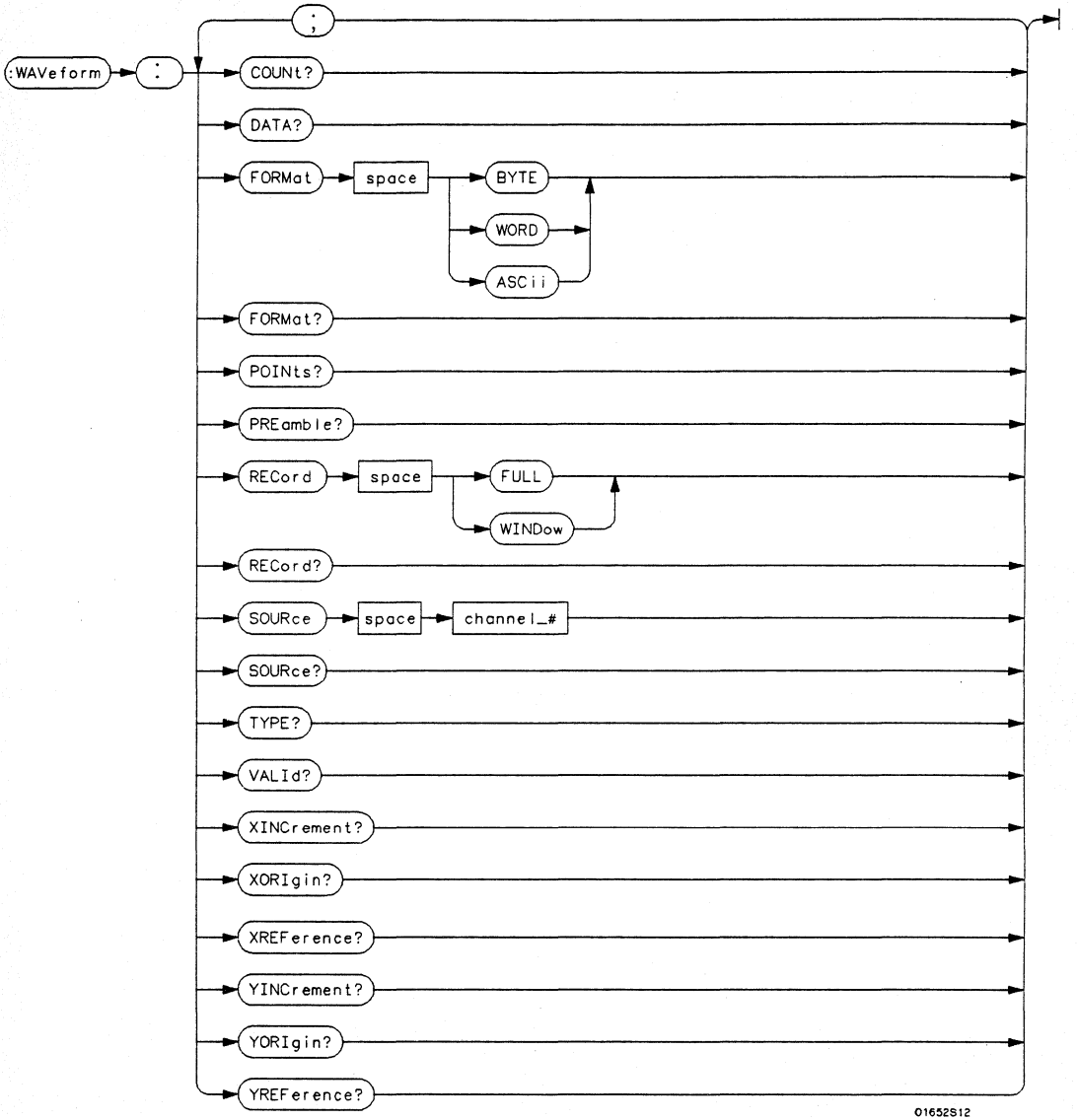
Example:
```
10 DIM Tr$[100]
20 OUTPUT XXX;":SCOPE:TIMEBASE:RANGE?"
30 ENTER XXX;Tr$
40 PRINT Tr$
50 END
```

WAVeform Subsystem 26

Introduction

The commands of the WAVeform subsystem are used to transfer waveform data from the oscilloscope to a controller. The commands are:

- COUNt
- DATA
- FORMat
- POINts
- PREamble
- RECord
- SOURce
- TYPe
- VALid
- XINCrement
- XORigin
- XREFerence
- YINCrement
- YORigin
- YREFerence



channel_# = {1|2}

Figure 26-1. WAVeform Subsystem Syntax Diagram

Waveform Record

The waveform record is actually contained in two portions; the waveform data and preamble. The waveform data is the actual data acquired for each point. The preamble contains the information for interpreting waveform data. Data in the preamble includes number of points acquired, format of acquired data, average count and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data for translation to time and voltage values.

The values set in the preamble are based on the settings of the variables in the ACQuire, WAVeform, CHANnel, and TIMebase subsystems. The ACQuire subsystem determines the acquisition type and the average count, the WAVeform subsystem sets the number of points and the format mode for sending waveform data over the remote interface and the CHANnel and TIMebase subsystems set all the X - Y parameters.

Data Acquisition Types

The two acquisition types that may be chosen are Normal and Average.

Normal Mode

In the Normal mode, with ACCumulate command OFF, the oscilloscope acquires waveform data and then displays the waveform. When the oscilloscope takes a new acquisition, the previously acquired waveform is erased from the display and replaced by the newly acquired waveform.

When ACCumulate is set ON, the oscilloscope displays all the waveform acquisitions without erasing the previously acquired waveform.

Average Mode

In the Average mode, the oscilloscope averages the data points on the waveform with previously acquired data. Averaging helps eliminate random noise from the displayed waveform. In this mode ACCumulate is set to OFF. When Average mode is selected the number of averages must also be specified using the COUNt command. Previously displayed waveform data is erased from the display and the newly averaged waveform is displayed.

Format for Data Transfer

There are three formats for transferring waveform data over the remote interface. The formats are WORD, BYTE, and ASCII.

WORD and BYTE formatted waveform records are transmitted using the arbitrary block program data format specified in IEEE-488.2. When you use this format, the ASCII character string "#8<DDDDDDDD>" is sent before the actual data. Each D represents an ASCII digit. The eight-digit number represents the number of bytes to follow.

For example, if 2048 points of data are to be transmitted, the ASCII string #800002048 would be sent.

BYTE Format

In BYTE format, the six least significant bits represent the waveform data. This means that the display is divided into 64 vertical increments. The most significant bit is not used. The second most significant bit is the overflow bit. If this bit is set to "1" and all data bits are set to "0" then the waveform is clipped at the top of the screen. If all "0"s are returned, then the waveform is clipped on the bottom of the display (see figure 26-2).

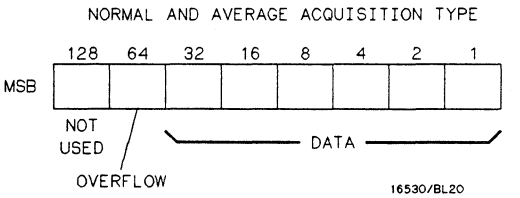


Figure 26-2. Byte Data Structure

The data returned in BYTE format are the same for either Normal or Average acquisition types. The data transfer rate in this format is faster than the other two formats.

WORD Format Word data is two bytes wide with the most significant byte of each word being transmitted first. Each 16-bit value effectively places a data point on screen. The screen therefore is divided into 16384 vertical increments. The WORD data structure for normal and average acquisition types are shown in figure 26-3.

The relationship between BYTE and WORD formats are similar. Byte data values equal word data values divided by 256. This is the reason that the least significant byte in the normal acquisition mode always contains "0"s. In the average acquisition mode, the extra bits of resolution gained by averaging occupy the least significant byte of the word. However, this is only true when RECord type is set to WINDow.

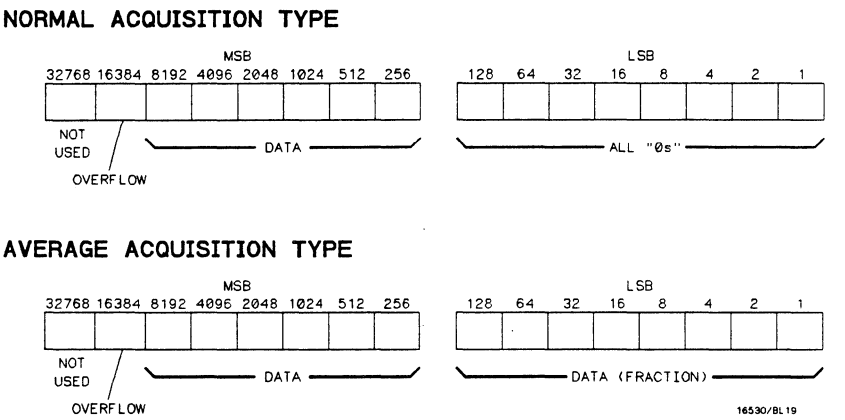


Figure 26-3. Word Data Structure

ASCII Format ASCII formatted waveform records are transmitted one value at a time, separated by a comma. The data values transmitted are the same as would be sent in the WORD format except that they are converted to an integer ASCII format (six or less characters) before being transmitted. The header before the data is not included in this format.

Data Conversion

Data sent from the HP 1652B/53B is raw data and must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble or by the queries of these values.

Conversion from Data Value to Voltage

The formula to convert a data value returned by the instrument to a voltage is:

voltage = [(data value − yreference) × yincrement] + yorigin

Conversion from Data Value to Time

The time value of a data point can be determined by the position of the data point. As an example, the third data point sent with XORIGIN = 16ns, XREFERENCE = 0 and XINCREMENT = 2ns. Using the formula:

time = [(data point number − xreference) × xincrement] + xorigin

would result in the following calculation:

time = [(3 - 0) × 2ns] + 16ns = 22ns.

Conversion from Data Value to Trigger Point

The trigger data point can be determined by calculating the closest data point to time 0.

WAVeform selector

The WAVeform selector is used as part of a compound command header to access the settings found in oscilloscope's Waveform menu. It always follows the SCOPe selector because it selects a branch below the SCOPe level in the command tree.

Command Syntax: :SCOPe:WAVeform

Example:

```
OUTPUT XXX; ":SCOPE:WAVEFORM:"
```

COUNt query

The COUNt query returns the AVERage count that was last specified in the Acquire subsystem. If the display mode is either NORMal or ACCumulate, a 1 is returned. If the display mode is AVERage, the average number is returned.

Query Syntax: :SCOPe:WAVeform:COUNt?

Returned Format: [:SCOPe:WAVeform:COUNt] <count> <NL>

where:

<count> ::= {2|4|8|16|32|64|128|256}

Example:

```
10 DIM Ac$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM COUNT?"
30 ENTER XXX;Ac$
40 PRINT Ac$
50 END
```

DATA query

The DATA query returns the waveform record stored in a specified channel buffer. The SOURce command of this subsystem has to be used to select the specified channel. The data is transferred based on the FORMAT (BYTE, WORD or ASCII) chosen and the RECORD specified (FULL or WINDOW). Since WAVeform:DATA is a query only, it can not be used to send a waveform record back to the oscilloscope from the controller. If a waveform record is to be saved for later reloading into the oscilloscope, the SYSTem:DATA command should be used. See the DATA instruction in the SYSTem subsystem for information concerning the < block data > parameter.

Query Syntax: :SCOPe:WAVeform:[SOURce CHANnel{1|2};]DATA?

Returned Format: [:SCOPe:WAVeform:DATA]#800004096 < block data > < NL >

The following example program moves data from the HP 1652B/53B to a controller.

Example:

```
100 CLEAR XXX
110 OUTPUT XXX;":SYSTEM:HEADER OFF"
120 OUTPUT XXX;":SCOPE:ACQUIRE:TYPE NORMAL"
130 OUTPUT XXX;":SCOPE:WAVEFORM:SOURCE CHANNEL1"
140 OUTPUT XXX;":SCOPE:WAVEFORM:FORMAT BYTE"
150 OUTPUT XXX;":SCOPE:WAVEFORM:RECORD FULL"
160 OUTPUT XXX;":SCOPE:AUTOSCALE"
170 DIM Header$[20]
180 Length=2048
190 ALLOCATE INTEGER WAVEFORM(1:Length)
200 OUTPUT XXX;":SCOPE:WAVEFORM:DATA?"
210 ENTER XXX USING "#,10A";Header$
220 ENTER XXX USING "#,B";Waveform(*)
230 ENTER XXX USING "#,B";Lastchar
240 END
```

FORMat command/query

The FORMat command specifies the data transmission mode of waveform data over the remote interface.

The query returns the currently specified format.

Command Syntax: :SCOPe:WAVeform:FORMat {BYTE|WORD|ASCii}

Example:
```
OUTPUT XXX;":SCOPE:WAV:FORMAT"
```

Query Syntax: :SCOPe:WAVeform:FORMat?"

Returned Format: [:SCOPe:WAVeform:FORMat] {BYTE|WORD|ASCii}<NL>

Example:
```
10 DIM Fo$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:FORMAT?"
30 ENTER XXX;Fo$
40 PRINT Fo$
50 END
```

POINts

query

When WAVeform RECord is set to FULL, the POINts query always returns a value of 2048 points. When WAVeform RECord is set to WINDow, then the query returns the number of points displayed on screen.

Query Syntax: :SCOPe:WAVeform:POINts?

Returned Format: [:SCOPe:WAVeform:POINts] <points> <NL>

where:

<points> ::= number of points depending on setting of WAVeform RECord command

Example:

```
10 DIM Po$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:POINTS?"
30 ENTER XXX;Po$
40 PRINT Po$
50 END
```

PREAmble query

The PREAmble query returns the preamble of the specified channel. The channel is specified using the SOURce command.

Note The short form for PREAMBLE is PREAmble. This is an intentional deviation from the normal truncation rule.

Query Syntax: :SCOPe:WAVeform:[SOURce CHANnel{1|2};]PREAmble?

Returned Format: [:SCOPe:WAVeform:PREAmble]

<format>,
<type>,
<points>,
<count>,
<Xincrement>,
<Xorigin>,
<Xreference>,
<Yincrement>,
<Yorigin>,
<Yreference><NL>

Example:

```
10 DIM Pr$[300]
20 OUTPUT XXX;":SCOPE:WAVEFORM:PREAMBLE?"
30 ENTER XXX;Pr$
40 PRINT Pr$
50 END
```

RECord command/query

The RECord command specifies the data you want to receive over the bus. The choices are FULL or WINDOW. When FULL is chosen the entire 2048 point record of the specified channel is transmitted over the bus. In WINDOW mode, only the data displayed on screen will be returned. Use the SOURce command to select the channel of interest. The query returns the present mode chosen.

Command Syntax: :SCOPe:WAVeform:RECord {FULL|WINDow}

Example:
```
OUTPUT XXX;":SCOPE:WAV:SOUR CHAN1:REC FULL"
```

Query Syntax: :SCOPe:WAVeform:RECord?

Returned Format: [:SCOPe:WAVeform:RECord] {FULL|WINDow}<NL>

Example:
```
10 DIM Wr$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:SOURCE CHANNEL1:RECORD?"
30 ENTER XXX;Wr$
40 PRINT Wr$
50 END
```

SOURce command/query

The SOURce command specifies the channel that is to be used for all subsequent waveform commands.

The query returns the presently selected channel.

Command Syntax: :SCOPe:WAVeform:SOURce CHANnel{1|2}

Example:
```
OUTPUT XXX;":SCOPE:WAVEFORM:SOURCE CHANNEL1"
```

Query Syntax: :SCOPe:WAVeform:SOURce?

Returned Format: [:SCOPe:WAVeform:SOURce] CHANnel<N><NL>

Example:
```
10 DIM Ws$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:SOURCE?"
30 ENTER XXX;Ws$
40 PRINT Ws$
50 END
```

TYPE query

The TYPE query returns the present acquisition type which was specified in the ACQuire subsystem.

Query Syntax: :SCOPe:WAVeform:TYPE?

Returned Format: [:SCOPe:WAVeform:TYPE]{NORmal|AVERage|ACCumulate}<NL>

Example:

```
10 DIM Wt$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:TYPE?"
30 ENTER XXX;Wt$
40 PRINT Wt$
50 END
```

VALid query

The VALid query checks the oscilloscope for acquired data. If a measurement is completed, and data has been acquired by all channels, then the query reports a 1. A 0 is reported if no data has been acquired for the last acquisition.

Query Syntax: :SCOPe:WAVeform:VALid?

Returned Format: [:SCOPe:WAVeform:VALid] {0|1}<NL>

where:

0 ::= No data acquired
1 ::= Data has been acquired

Example:

```
10 DIM Da$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:VALID?"
30 ENTER XXX;Da$
40 PRINT Da$
50 END
```

XINCrement

query

The XINCrement query returns the X-increment currently in the preamble. This value is the time between the consecutive data points.

Query Syntax: :SCOPe:WAVeform:XINCrement?

Returned Format: [:SCOPe:WAVeform:XINCrement] <value> <NL>

where:

<value> ::= X-increment value currently in preamble

Example:

```
10 DIM Xi$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:XINCREMENT?"
30 ENTER XXX;Xi$
40 PRINT Xi$
50 END
```

XORigin query

The XORigin query returns the X-origin value currently in the preamble. The value represents the time of the first data point in memory with respect to the trigger point.

Query Syntax: :SCOPe:WAVeform:XORigin?

Returned Format: [:SCOPe:WAVeform:XORigin] <value> <NL>

where:

<value> ::= X-origin value currently in preamble

Example:

```
10 DIM Xo$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:XORigin?"
30 ENTER XXX;Xo$
40 PRINT Xo$
50 END
```

XREFerence query

The XREFerence query returns the X-reference value in the preamble. The value specifies the first data point in memory and is always 0.

Query Syntax: :SCOPe:WAVeform:XREFerence?

Returned Format: [:SCOPe:WAVeform:XREFerence] <value> <NL>

where:

<value> ::= X-reference value in preamble

Example:

```
10 DIM Xo$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:XREFerence?"
30 ENTER XXX;Xo$
40 PRINT Xo$
50 END
```

YINCrement query

The YINCrement query returns the Y-increment currently in the preamble. This value is the voltage difference between consecutive data values.

Query Syntax: :SCOPe:WAVeform:YINCrement?

Returned Format: [:SCOPe:WAVeform:YINCrement] <value> <NL>

where:

<value> ::= Y-increment value currently in preamble

Example:

```
10 DIM Yi$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:YINCREMENT?"
30 ENTER XXX;Yi$
40 PRINT Yi$
50 END
```

YORigin query

The YORigin query returns the Y-origin value currently in the preamble. This value is the voltage at the center of the screen.

Query Syntax: :SCOPe:WAVeform:YORigin?

Returned Format: [:SCOPe:WAVeform:YORigin] <value> <NL>

where:

<value> ::= Y-origin value currently in preamble

Example:

```
10 DIM Yo$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:YORigin?"
30 ENTER XXX;Yo$
40 PRINT Yo$
50 END
```

YREFerence query

The YREFerence query returns the Y-reference value in the preamble. The value specifies the data value at center screen where Y-origin occurs.

Query Syntax: :SCOPe:WAVeform:YREFerence?

Returned Format: [:SCOPe:WAVeform:YREFerence] <value> <NL>

where:

<value> ::= Y-reference value in preamble

Example:

```
10 DIM Yo$[100]
20 OUTPUT XXX;":SCOPE:WAVEFORM:YREFerence?"
30 ENTER XXX;Yo$
40 PRINT Yo$
50 END
```

MEASure Subsystem 27

Introduction

The instructions in the MEASure subsystem are used to make automatic parametric measurements on displayed waveforms. The instructions are:

- ALL
- FALLTime
- FREQuency
- NWIDth
- OVERShoot
- PERiod
- PRESHoot
- PWIDth
- RISETime
- SOURce
- VAMPlitude
- VBASe
- VMAX
- VMIN
- VPP
- VTOP

Before using any of the MEASure subsystem queries, be sure that you have used to SOURce command to specify which channel is to be used. All subsequent measurements will be made from that channel's waveform.

If a waveform characteristic cannot be measured, the instrument responds with 9.9E+37.

The following characteristics can be measured:

Frequency The frequency of the first complete cycle displayed is measured using the 50% level.

Period The period of the first displayed waveform is measured at the 50% level.

Peak-to-Peak The absolute minimum and the maximum voltages for the selected source are measured.

Positive Pulse Width Pulse width is measured at the 50% level of the first displayed pulse.

Negative Pulse Width Pulse width is measured at the 50% level of the first displayed pulse.

Risetime The risetime of the first displayed rising edge is measured. To obtain the best possible measurement accuracy, select the fastest sweep speed while keeping the rising edge on the display. The risetime is determined by measuring time at the 10% and the 90% voltage points of the rising edge.

Falltime Falltime is measured between the 10% and the 90% points of the first displayed falling edge. To obtain the best possible measurement accuracy, select the fastest sweep speed possible while keeping the falling edge on the display.

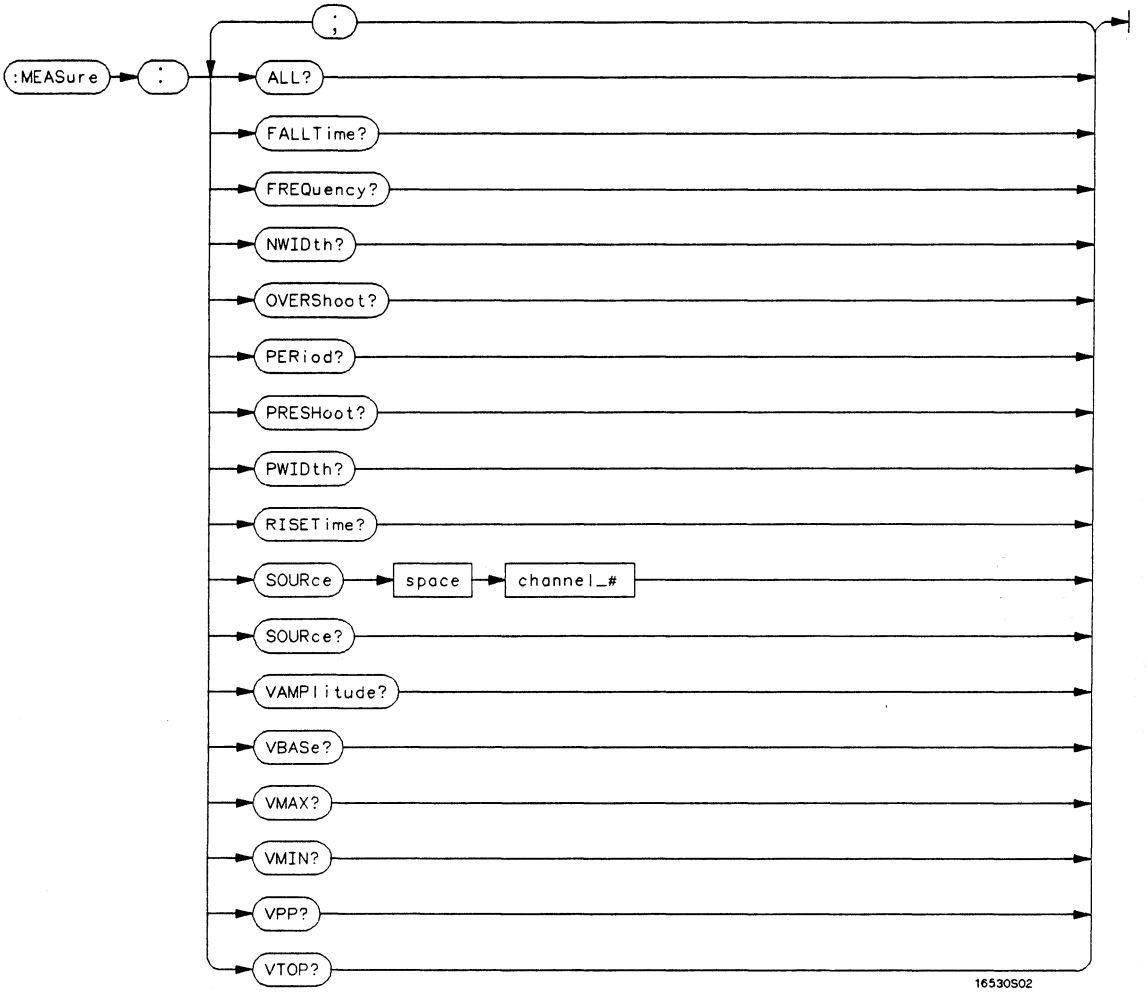
Preshoot and Overshoot Preshoot and overshoot measure the perturbation on a waveform above or below the top and base voltages.

Preshoot is a perturbation before a rising or a falling edge and measured as a percentage of the top-base voltage.

Overshoot is a perturbation after a rising or falling edge and is measured as a percentage of the top-base voltage.

For complete details of the measurement algorithms, refer to the *Front-panel Operating Reference Manual.*

Refer to figure 27-1 for the MEASure subsystem syntax diagram.



channel_# = *an integer* {*1* | *2*}.

Figure 27-1. MEASure Subsystem Syntax Diagram

MEASure selector

The MEASure selector is used as part of a compound command header to access the settings found in oscilloscope's Measure menu. It always follows the SCOPe selector because it selects a branch below the SCOPe level in the command tree.

Command Syntax: :SCOPe:MEASure

Example:

```
OUTPUT XXX; ":SCOPE:MEASURE:SOURCE CHAN2"
```

Note

All queries in this subsystem return the measurement results of the last channel specified by the SOURce command. If you want measurement results from the other channel, you must use the SOURce command before using any of the queries.

ALL

query

The ALL query makes a set of measurements on the displayed waveform using the selected source.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]ALL?

Returned Format:
```
[:SCOPe:MEASure:ALL PERiod] <real number>;
[RISETime] <real number>;
[FALLTime] <real number>;
[FREQuency] <real number>;
[PWIDtH] <real number>;
[NWIDtH] <real number>;
[VPP] <real number>;
[VAMPlitude] <real number>;
[PRESHoot] <real number>;
[OVERShoot] <real number><NL>
```

Example:
```
10 DIM Query$[300]
20 !PRINTER IS 701 !THIS LINE SENDS RESULTS TO PRINTER
30 OUTPUT XXX;":SCOPE:MEASURE:SOUR CHAN1"
40 OUTPUT XXX;":SCOPE:MEASURE:ALL?"
50 ENTER XXX;Query$
60 Query$=Query$[POS(Query$,";")+1]
70 LOOP
80 I=POS(Query$,";")
90 EXIT IF NOT I
100 PRINT Query$[1,I-1]
110 Query$=Query$[I+1]
120 END LOOP
130 PRINT Query$
140 PRINTER IS 1
150 END
```

FALLTime

query

The FALLTime query makes a fall time measurement on the selected channel. The measurement is made between the 90% to the 10% voltage point of the first falling edge displayed on screen.

Note The short form of FALLTIME is FALLTime. This is an intentional deviation of the normal truncation rule.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]FALLTime?

Returned Format: [:SCOPe:MEASure:FALLTime] <value><NL>

where:

<value> ::= time in seconds between 10% and 90% voltage points

Example:

```
10 DIM Ft$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHANNEL2;FALLTIME?"
30 ENTER XXX;Ft$
40 PRINT Ft$
50 END
```

FREQuency query

The FREQency query makes a frequency measurement on the selected channel. The measurement is made using the first complete displayed cycle at the 50% voltage level.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]FREQuency?

Returned Format: [:MEAsure:FREQuency] <value><NL>

where:

<value> ::= frequency in Hertz

Example:

```
10 DIM Frcy$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOUR CHAN1;FREQ?"
30 ENTER XXX;Frcy$
40 PRINT Frcy$
50 END
```

NWIDth query

The NWIDth query makes a negative width time measurement on the selected channel. The measurement is made between the 50% points of the first falling and the next rising edge displayed on screen.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]NWIDth?

Returned Format: [:SCOPe:MEASure:NWIDth] <value><NL>

where:

<value> ::= negative pulse width in seconds

Example:

```
10 DIM Nw$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHAN2;NWID?"
30 ENTER XXX;Nw$
40 PRINT Nw$
50 END
```

OVERShoot

query

The OVERShoot query makes an overshoot measurement on the selected channel. The measurement is made by finding a distortion following the first major transition. The result is the ratio of VMAX or VMIN vs. VAMPlitude.

Note The short form of OVERSHOOT is OVERShoot. This is an intentional deviation from the normal truncation rule.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]OVERShoot?

Returned Format: [:SCOPe:MEASure:OVERShoot] <value><NL>

where:

<value> ::= ratio of overshoot to Vamplitude

Example:

```
10 DIM Ovs$[100]
20 OUTPUT XXX;":SCOPE:MEASURE SOURCE CHAN1;OVER?"
30 ENTER XXX;Ovs$
40 PRINT Ovs$
50 END
```

PERiod query

The PERiod query makes a period measurement on the selected channel. The measurement equivalent to the inverse of frequency.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]PERiod?

Returned Format: [:SCOPe:MEASure:PERiod] <value><NL>

where:

<value> ::= waveform period in seconds

Example:

```
10 DIM Pd$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHANNEL1;PERIOD?"
30 ENTER XXX;Pd$
40 PRINT Pd$
50 END
```

PRESHoot query

The PRESHoot query makes the preshoot measurement on the selected channel. The measurement is made by finding a distortion which precedes the first major transition on screen. The result is the ratio of VMAX or VMIN vs. VAMPlitude.

Note The short form of PRESHOOT is PRESHoot. This is an intentional deviation of the normal truncation rule.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]PRESHoot?

Returned Format: [:SCOPe:MEASure:PRESHoot] <value><NL>

where:

<value> ::= ratio of preshoot to Vamplitude

Example:

```
10 DIM Prs$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:CHANNEL2;PRESH?"
30 ENTER XXX;Prs$
40 PRINT Prs$
50 END
```

PWIDth query

The PWIDth query makes a positive pulse width measurement on the selected channel. The measurement is made by finding the time difference between the 50% points of the first rising and the next falling edge displayed on screen.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]PWIDth?

Returned Format: [:SCOPe:MEASure:PWIDth] <value><NL>

where:

<value> ::= positive pulse width in seconds

Example:

```
10 DIM Pw$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHANNEL2;PWIDTH?"
30 ENTER XXX;Pw$
40 PRINT Pw$
50 END
```

RISETime

query

The RISETime query makes a risetime measurement on the selected channel by finding the 10% and 90% voltage levels of the first rising edge displayed on screen.

Note 

The short form of RISETIME is RISETime. This is an intentional deviation from the normal truncation rule.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]RISETime?

Returned Format: [:SCOPe:MEASure:RISETime] <value><NL>

where:

<value> ::= risetime in seconds

Example:

```
10 DIM Tr$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHANNEL1;RISETIME?"
30 ENTER XXX;Tr$
40 PRINT Tr$
50 END
```

SOURce command/query

The SOURce command specifies the source to be used for subsequent measurements. If the source is not specified, the last waveform source is assumed.

The query returns the presently specified channel.

Command Syntax: :SCOPe:MEASure:SOURce <source>

where:

<source> ::= {1 | 2}

Example:
```
OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHAN1"
```

Query Syntax: :SCOPe:MEASure:SOURce?

Returned Format: [:SCOPe:MEASure:SOURce] CHANnel<N><NL>

Example:
```
10 DIM So$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE?"
30 ENTER XXX;So$
40 PRINT So$
50 END
```

VAMPlitude query

The VAMPlitude query makes a voltage measurement on the selected channel. The measurement is made by finding the relative maximum and minimum points on screen.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]VAMPlitude?

Returned Format: [:SCOPe:MEASure:VAMPlitude] <value><NL>

where:

<value> ::= difference between top and base voltage

Example:

```
10 DIM Va$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHANNEL2;VAMP?"
30 ENTER XXX;Va$
40 PRINT Va$
50 END
```

VBASe query

The VBASe query returns the base voltage (relative minimum) of a displayed waveform. The measurement is made on the selected source.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]VBASe?

Returned Format: [:SCOPe:MEASure:VBASe] <value><NL>

where:

<value> ::= voltage at base level of selected waveform

Example:

```
10 DIM Vb$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHAN1;VBAS?"
30 ENTER XXX;Vb$
40 PRINT Vb$
50 END
```

VMAX

query

The VMAX query returns the absolute maximum voltage of the selected source.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]VMAX?

Returned Format: [:SCOPe:MEASure:VMAX] <value><NL>

where:

<value> ::= maximum voltage of selected waveform

Example:

```
10 DIM Vma$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHAN2;VMAX?"
30 ENTER XXX;Vma$
40 PRINT Vma$
50 END
```

VMIN query

The VMIN query returns the absolute minimum voltage present on the selected source.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]VMIN?

Returned Format: [:SCOPe:MEASure VMIN] <value><NL>

where:

<value> ::= minimum voltage of selected waveform

Example:

```
10 DIM Vmi$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHAN1;VMIN?"
30 ENTER XXX;Vmi$
40 PRINT Vmi$
50 END
```

VPP

query

The VPP query makes a peak-to-peak voltage measurement on the selected source. The measurement is made by finding the absolute maximum and minimum points on the displayed waveform.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]VPP?

Returned Format: [:SCOPe:MEASure:VPP] <value><NL>

where:

<value> ::= peak to peak voltage of selected waveform

Example:

```
10 DIM Vpp$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHAN1;VPP?"
30 ENTER XXX;Vpp$
40 PRINT Vpp$
50 END
```

VTOP query

The VTOP query returns the voltage at the top (relative maximum) of waveform on the selected source.

Query Syntax: :SCOPe:MEASure:[SOURce CHANnel{1|2};]VTOP?

Returned Format: [:SCOPe:MEASure:VTOP] <value><NL>

where:

<value> ::= voltage at the top of the selected waveform

Example:

```
10 DIM Vt$[100]
20 OUTPUT XXX;":SCOPE:MEASURE:SOURCE CHAN2;VTOP?"
30 ENTER XXX;Vt$
40 PRINT Vt$
50 END
```

Message Communication and System Functions

A

Introduction

This appendix describes the operation of instruments that operate in compliance with the IEEE 488.2 (syntax) standard. Although the HP 1652B and HP 1653B logic analyzers are RS-232C instruments, they were designed to be compatible with other Hewlett-Packard IEEE 488.2 compatible instruments.

The IEEE 488.2 standard is a new standard. Instruments that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (HP-IB bus standard); however, IEEE 488.1 compatible instruments may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the instrument and the controller will communicate. It also defines some common capabilities, which are found in all IEEE 488.2 instruments. This appendix also contains a few items which are not specifically defined by IEEE 488.2, but deal with message communication or system functions.

Note 

The syntax and protocol for RS-232C program messages and response messages for the HP 1652B/1653B are structured very similar to those described by 488.2. In most cases, the same structure shown in this appendix for 488.2 will also work for RS-232C. Because of this, no additional information has been included for RS-232C.

Protocols

The protocols of IEEE 488.2 define the overall scheme used by the controller and the instrument to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

Functional Elements

Before proceeding with the description of the protocol, a few system components should be understood.

Input Buffer. The input buffer of the instrument is the memory area where commands and queries are stored prior to being parsed and executed. It allows a controller to send a string of commands to the instrument which could take some time to execute, and then proceed to talk to another instrument while the first instrument is parsing and executing commands.

Output Queue. The output queue of the instrument is the memory area where all output data (< response messages >) are stored until read by the controller.

Parser. The instrument's parser is the component that interprets the commands sent to the instrument and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and executing of commands begins when either the instrument recognizes a < program message terminator > (defined later in this appendix) or the input buffer becomes full. If you wish to send a long sequence of commands to be executed and then talk to another instrument while they are executing, you should send all the commands before sending the < program message terminator >.

Protocol Overview

The instrument and controller communicate using < program message > s and < response message > s. These messages serve as the containers into which sets of program commands or instrument responses are placed. < program message > s are sent by the controller to the instrument, and < response message > s are sent from the instrument to the controller in response to a query message. A < query message > is defined as being a < program message > which contains one or more queries. The instrument will only talk when it has received a valid query message, and therefore has something to say. The controller should only attempt to read a response after sending a complete query message, but before sending another < program message > . The basic rule to remember is that the instrument will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

When the instrument is turned on, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

The instrument and the controller communicate by exchanging complete < program message > s and < response message > s. This means that the controller should always terminate a < program message > before attempting to read a response. The instrument will terminate < response message > s except during a hardcopy output.

If a query message is sent, the next message passing over the bus should be the < response message > . The controller should always read the complete < response message > associated with a query message before sending another < program message > to the same instrument.

The instrument allows the controller to send multiple queries in one query message. This is referred to as sending a "compound query." As will be noted later in this appendix, multiple queries in a query message are separated by semicolons. The responses to each of the queries in a compound query will also be separated by semicolons.

Commands are executed in the order they are received.

Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner. Some of the protocol exceptions are shown below.

Command Error. A command error will be reported if the instrument detects a syntax error or an unrecognized command header.

Execution Error. An execution error will be reported if a parameter is found to be out of range, or if the current settings do not allow execution of a requested command or query.

Device-specific Error. A device-specific error will be reported if the instrument is unable to execute a command for a strictly device dependent reason.

Query Error. A query error will be reported if the proper protocol for reading a query is not followed. This includes the interrupted and unterminated conditions described in the following paragraphs.

Syntax Diagrams

The syntax diagrams in this appendix are similar to the syntax diagrams in the IEEE 488.2 specification. Commands and queries are sent to the instrument as a sequence of data bytes. The allowable byte sequence for each functional element is defined by the syntax diagram that is shown with the element description.

The allowable byte sequence can be determined by following a path in the syntax diagram. The proper path through the syntax diagram is any path that follows the direction of the arrows. If there is a path around an element, that element is optional. If there is a path from right to left around one or more elements, that element or those elements may be repeated as many times as desired.

Syntax Overview

This overview is intended to give a quick glance at the syntax defined by IEEE 488.2. It should allow you to understand many of the things about the syntax you need to know. This appendix also contains the details of the IEEE 488.2 defined syntax.

IEEE 488.2 defines the blocks used to build messages which are sent to the instrument. A whole string of commands can therefore be broken up into individual components.

Figure A-1 shows a breakdown of an example <program message>. There are a few key items to notice:

1. A semicolon separates commands from one another. Each <program message unit> serves as a container for one command. The <program message unit>s are separated by a semicolon.

2. A <program message> is terminated by a <NL> (new line). The recognition of the <program message terminator>, or <PMT>, by the parser serves as a signal for the parser to begin execution of commands. The <PMT> also affects command tree traversal (see the Programming and Documentation Conventions chapter).

3. Multiple data parameters are separated by a comma.

4. The first data parameter is separated from the header with one or more spaces.

5. The header MACHINE1:ASSIGN 2,3 is an example of a compound header. It places the parser in the machine subsystem until the <NL> is encountered.

6. A colon preceding the command header returns you to the top of the command tree.

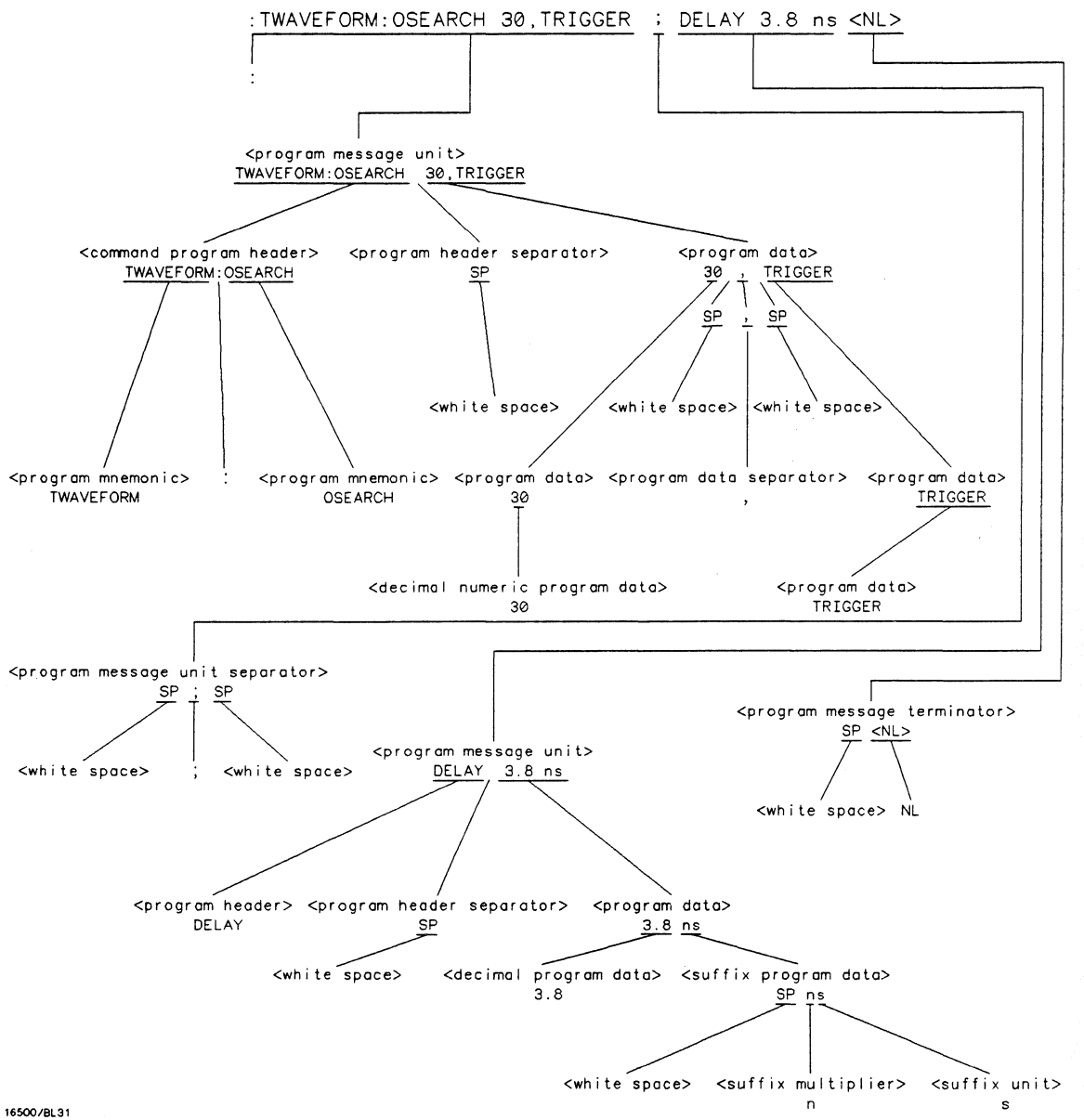


Figure A-1. < program message > Parse Tree

Device Listening Syntax

The listening syntax of IEEE 488.2 is designed to be more forgiving than the talking syntax. This allows greater flexibility in writing programs, as well as allowing them to be easier to read.

Upper/Lower Case Equivalence. Upper and lower case letters are equivalent. The mnemonic SINGLE has the same semantic meaning as the mnemonic single.

<white space>. <white space> is defined to be one or more characters from the ASCII set of 0 - 32 decimal, excluding 10 decimal (NL). <white space> is used by several instrument listening components of the syntax. It is usually optional, and can be used to increase the readability of a program.

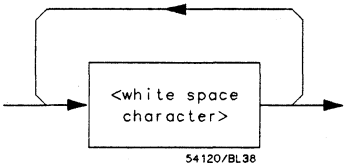


Figure A-2. <white space>

< program message >. The < program message > is a complete message to be sent to the instrument. The instrument will begin executing commands once it has a complete < program message >, or when the input buffer becomes full. The parser is also repositioned to the root of the command tree after executing a complete < program message >. Refer to "Tree Traversal Rules" in the "Programming and Documentation Conventions," chapter 4 for more details.

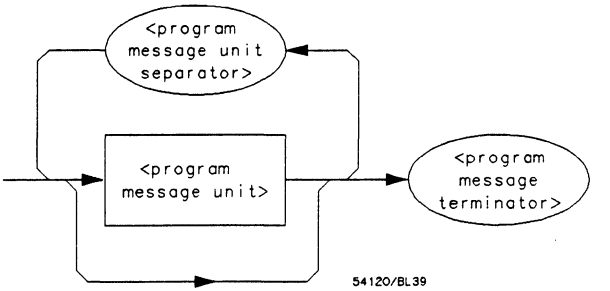


Figure A-3. < program message >

< program message unit >. The < program message unit > is the container for individual commands within a < program message >.

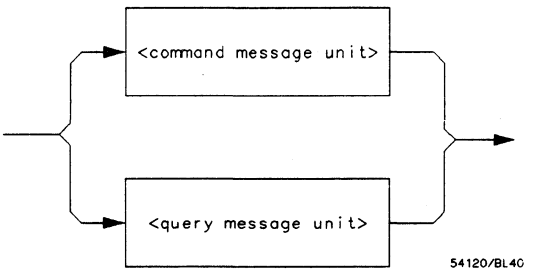


Figure A-4. < program message unit >

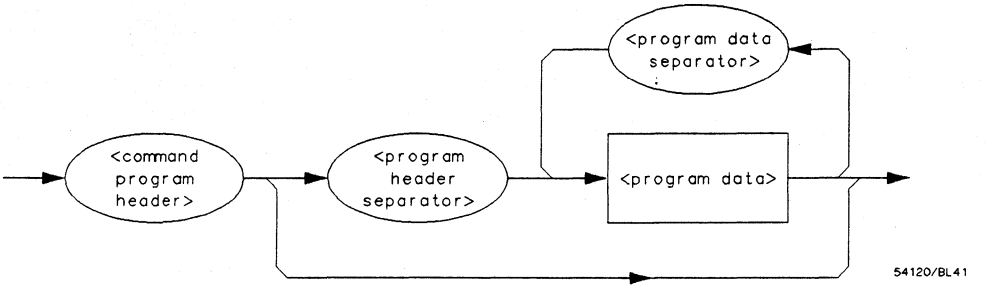


Figure A-5. < command message unit >

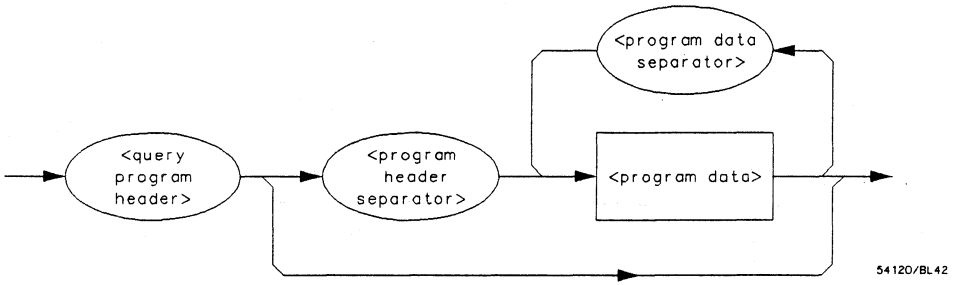


Figure A-6. < query message unit >

< program message unit separator >. A semicolon separates < program message unit > s, or individual commands.

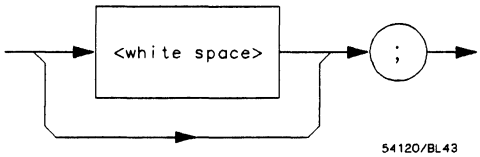


Figure A-7. < program message unit separator >

< command program header > / < query program header >. These elements serve as the headers of commands or queries. They represent the action to be taken.

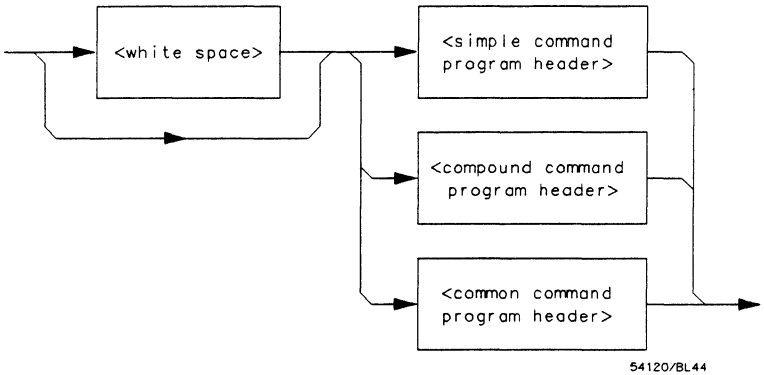
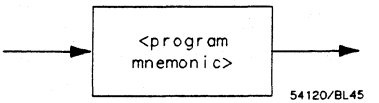
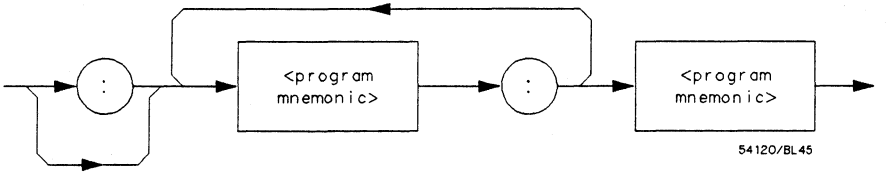


Figure A-8. < command program header >

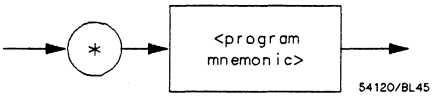
Where <simple command program header> is defined as



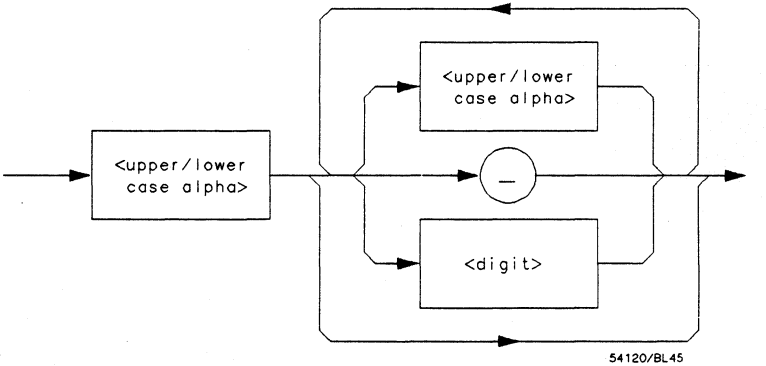
Where <compound command program header> is defined as



Where <common command program header> is defined as



Where <program mnemonic> is defined as

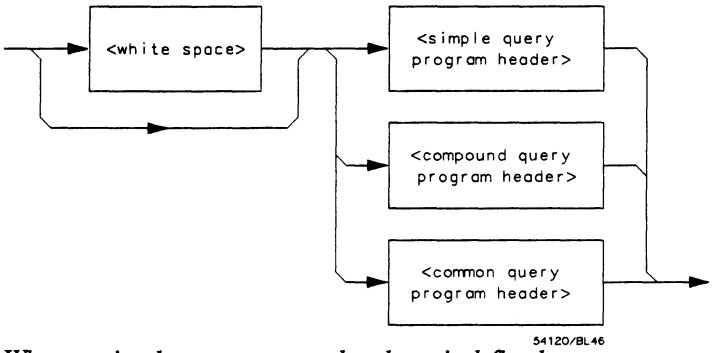


Where <upper/lower case alpha> is defined as a single ASCII encoded byte in the range 41 - 5A, 61 - 7A (65 - 90, 97 - 122 decimal).

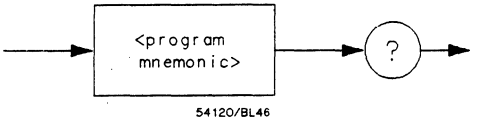
Where <digit> is defined as a single ASCII encoded byte in the range 30 - 39 (48 - 57 decimal).

Where (_) represents an "underscore", a single ASCII-encoded byte with the value 5F (95 decimal).

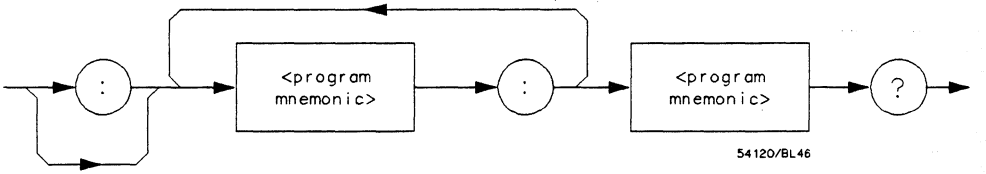
Figure A-8. <command program header> (continued)



Where <simple query program header> is defined as



Where <compound query program header> is defined as



Where <common query program header> is defined as

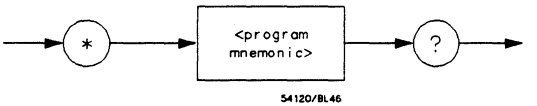


Figure A-9. <query program header>

< program data >. The < program data > element represents the possible types of data which may be sent to the instrument. The HP 1652B/1653B will accept the following data types: < character program data >, < decimal numeric program data >, < suffix program data >, < string program data >, and < arbitrary block program data >.

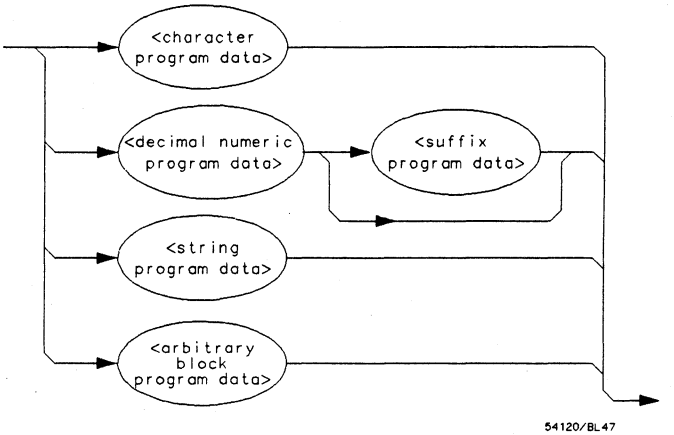


Figure A-10. < program data >

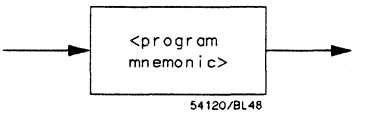
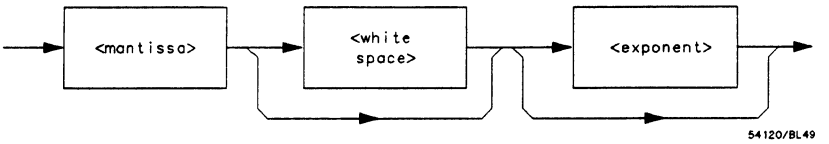
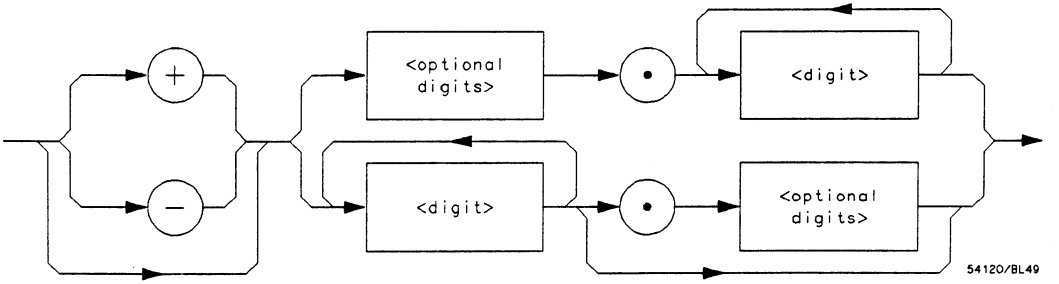


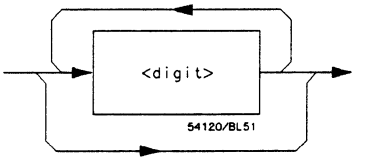
Figure A-11. < character program data >



Where <mantissa> is defined as



Where <optional digits> is defined as



Where <exponent> is defined as

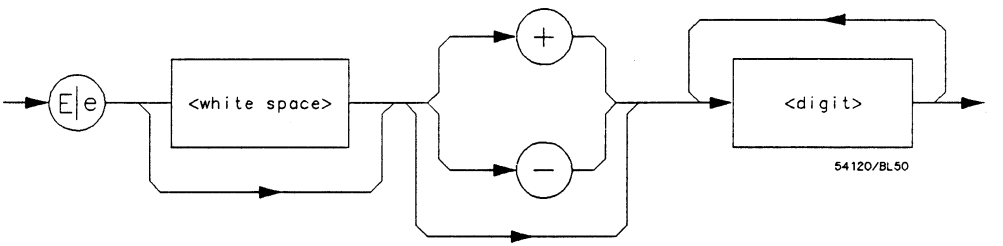


Figure A-12. <decimal numeric program data>

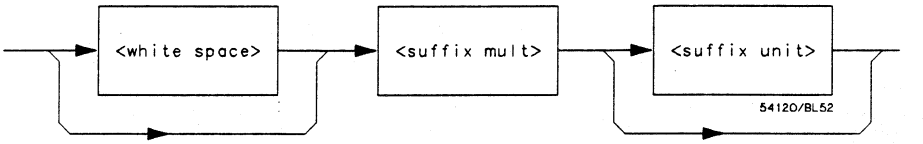


Figure A-13. < suffix program data >

Suffix Multiplier. The suffix multipliers that the instrument will accept are shown in table A-1.

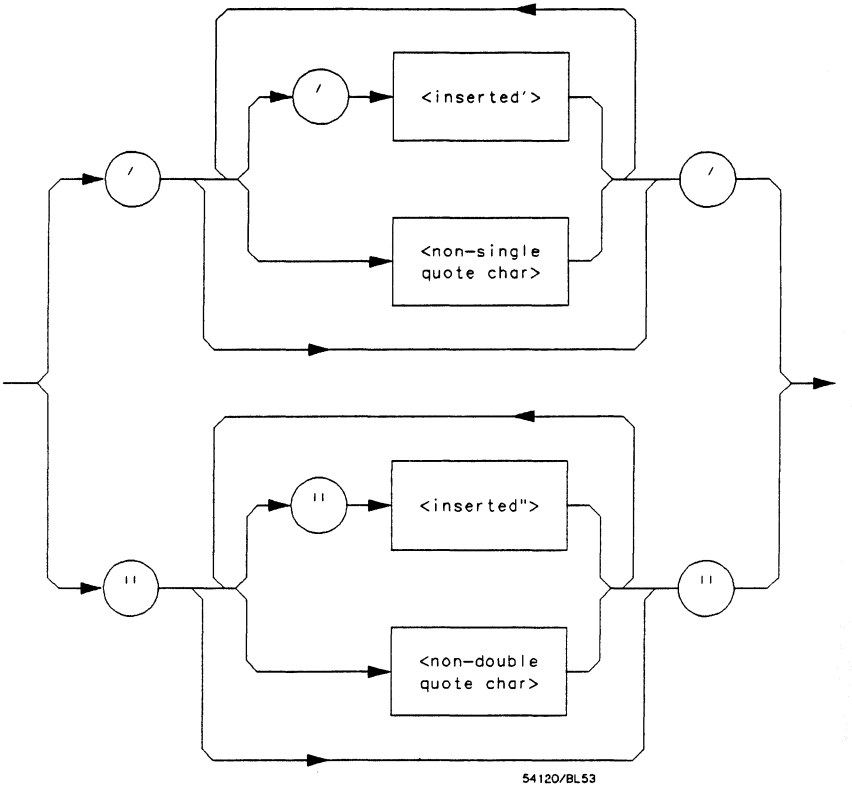
Table A-1. < suffix mult >

Value	Mnemonic
1E18	EX
1E15	PE
1E12	T
1E9	G
1E6	MA
1E3	K
1E-3	M
1E-6	U
1E-9	N
1E-12	P
1E-15	F
1E-18	A

Suffix Unit. The suffix units that the instrument will accept are shown in table A-2.

Table A-2. < suffix unit >

Suffix	Referenced Unit
V	Volt
S	Second



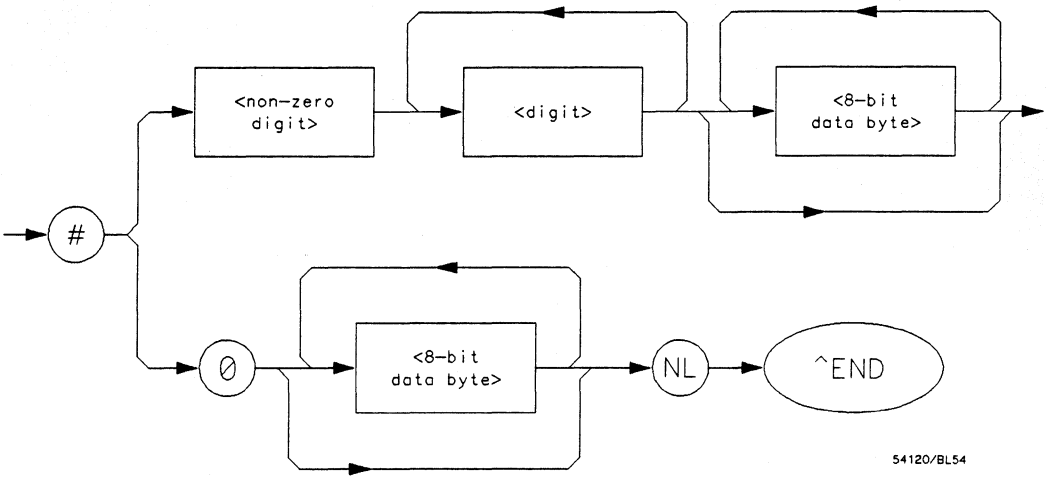
Where <inserted '> is defined as a single ASCII character with the value 27 (39 decimal).

Where <non-single quote char> is defined as a single ASCII character of any value except 27 (39 decimal).

Where <inserted "> is defined as a single ASCII character with the value 22 (34 decimal).

Where <non-double quote char> is defined as a single ASCII character of any value except 22 (34 decimal)

Figure A-14. < string program data >



Where <non-zero digit> is defined as a single ASCII encoded byte in the range 31 - 39 (49 - 57 decimal).

Where <8-bit byte> is defined as an 8-bit byte in the range 00 - FF (0 - 255 decimal).

Figure A-15. <arbitrary block program data>

<program data separator>. A comma separates multiple data parameters of a command from one another.

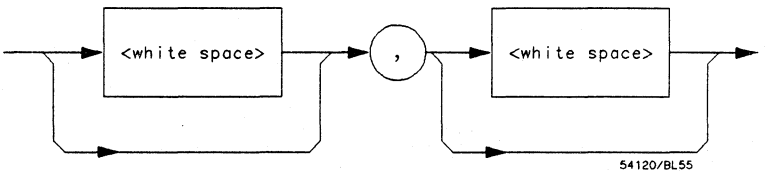


Figure A-16. <program data separator>

< program header separator >. A space separates the header from the first or only parameter of the command.

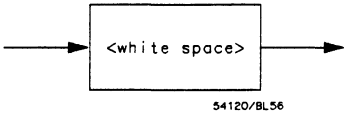
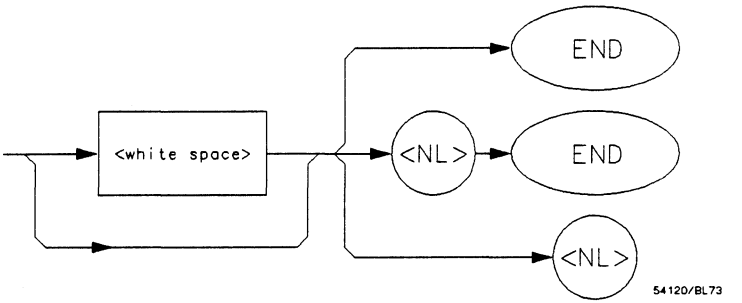


Figure A-17. < program header separator >

< program message terminator >. The < program message terminator > or < PMT > serves as the terminator to a complete < program message >. When the parser sees a complete < program message > it will begin execution of the commands within that message. The < PMT > also resets the parser to the root of the command tree.



Where < NL > is defined as a single ASCII-encoded byte 0A (10 decimal).

Figure A-18. < program message terminator >

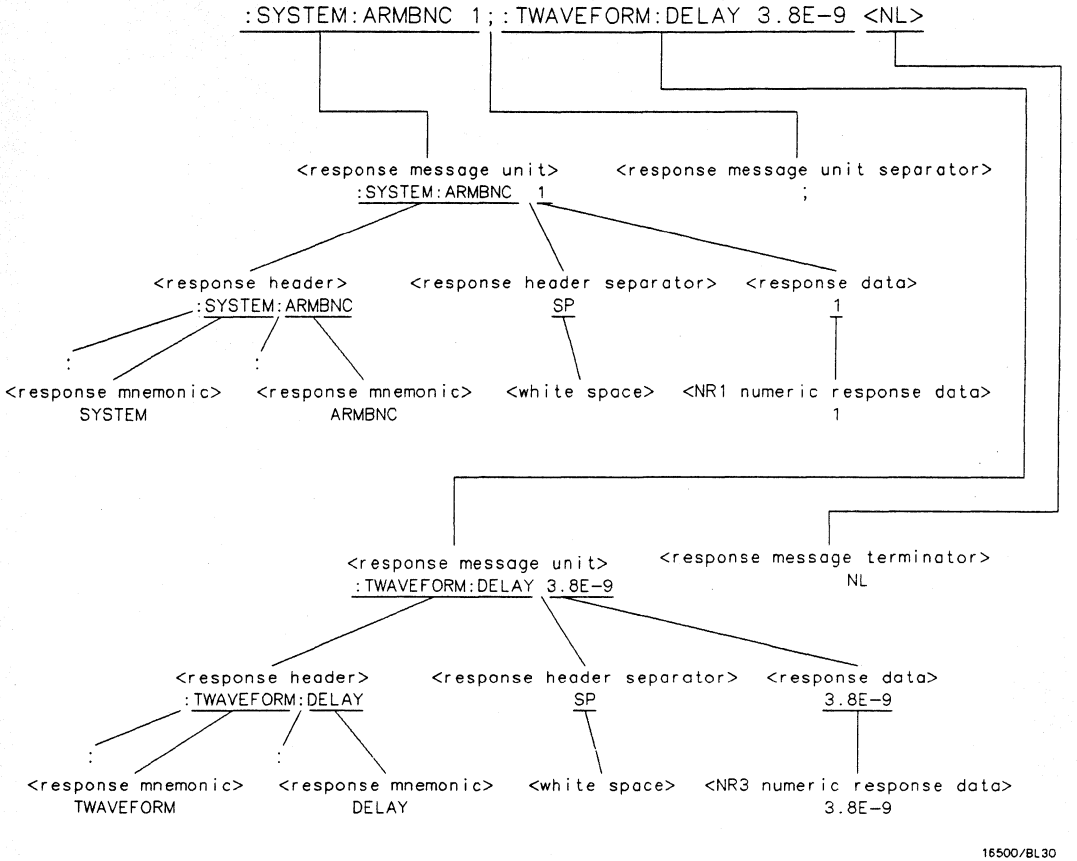


Figure A-19. <response message> Tree

Device Talking Syntax

The talking syntax of IEEE 488.2 is designed to be more precise than the listening syntax. This allows the programmer to write routines which can easily interpret and use the data the instrument is sending. One of the implications of this is the absence of < white space > in the talking formats. The instrument will not pad messages which are being sent to the controller with spaces.

< response message >. This element serves as a complete response from the instrument. It is the result of the instrument executing and buffering the results from a complete < program message >. The complete < response message > should be read before sending another < program message > to the instrument.

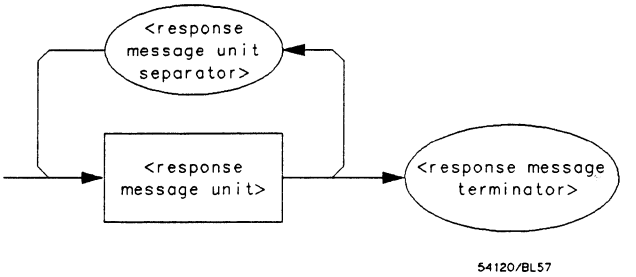
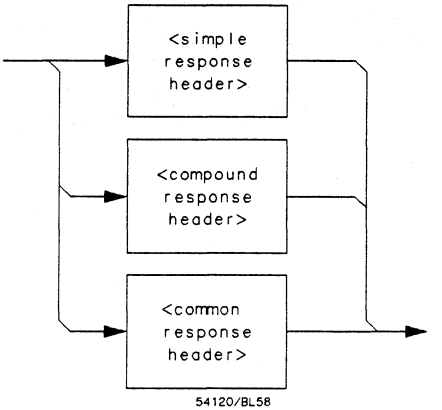


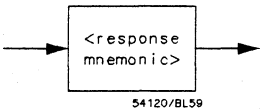
Figure A-20. < response message >

< response message unit >. This element serves as the container of individual pieces of a response. Typically a < query message unit > will generate one < response message unit >, although a < query message unit > may generate multiple < response message unit > s.

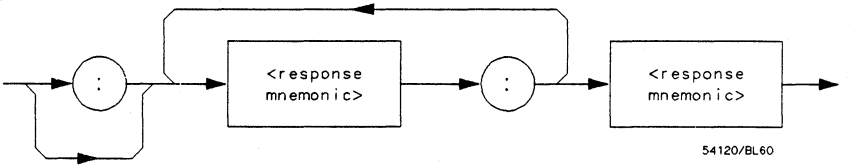
< response header >. The < response header >, when returned, indicates what the response data represents.



Where <simple response mnemonic> is defined as



Where <compound response header> is defined as



Where <common response header> is defined as

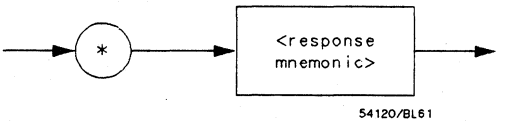
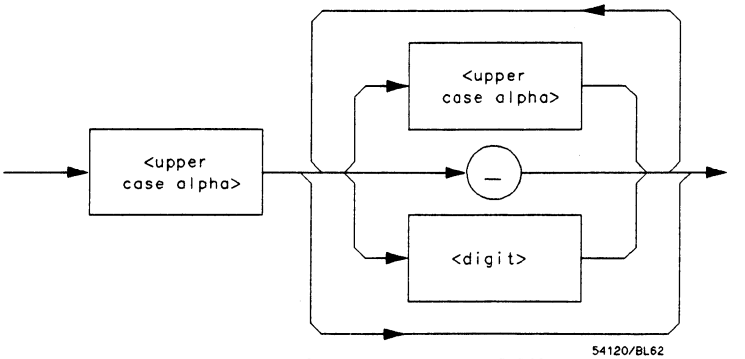


Figure A-21. <response message unit>

Where <response mnemonic> is defined as



Where <uppercase alpha> is defined as a single ASCII encoded byte in the range 41 - 5A (65 - 90 decimal).

Where (_) represents an "underscore", a single ASCII-encoded byte with the value 5F (95 decimal).

Figure A-21. <response message unit> (Continued)

<response data>. The <response data> element represents the various types of data which the instrument may return. These types include: <character response data>, <nr1 numeric response data>, <nr3 numeric response data>, <string response data>, <definite length arbitrary block response data>, and <arbitrary ASCII response data>.

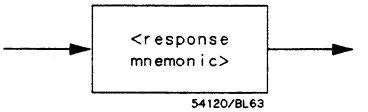


Figure A-22. <character response data>

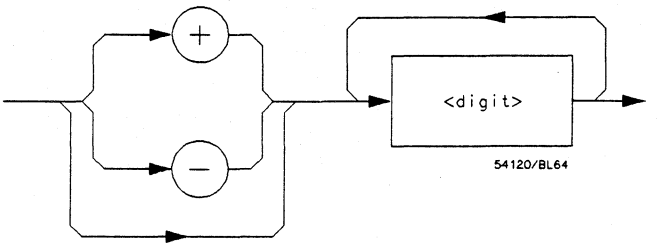


Figure A-23. < nr1 numeric response data >

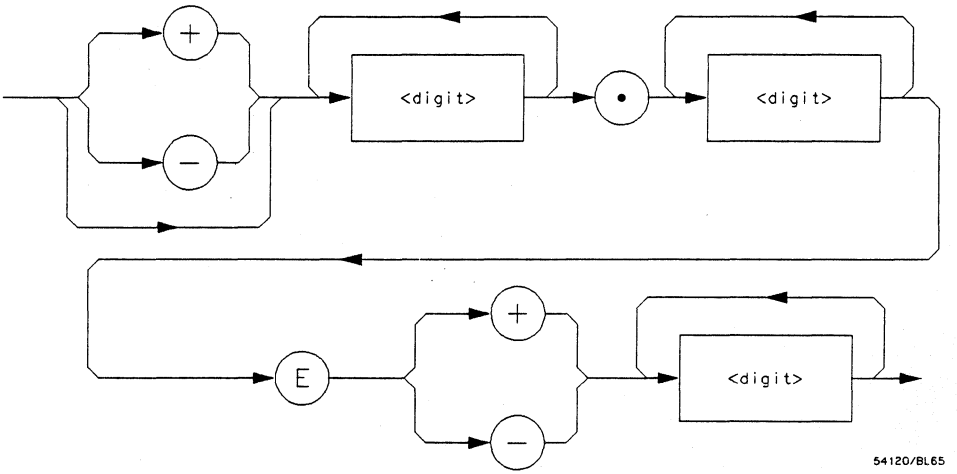


Figure A-24. < nr3 numeric response data >

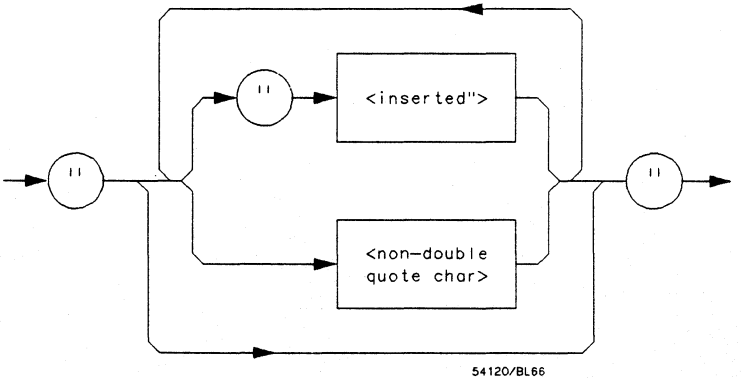


Figure A-25. < string response data >

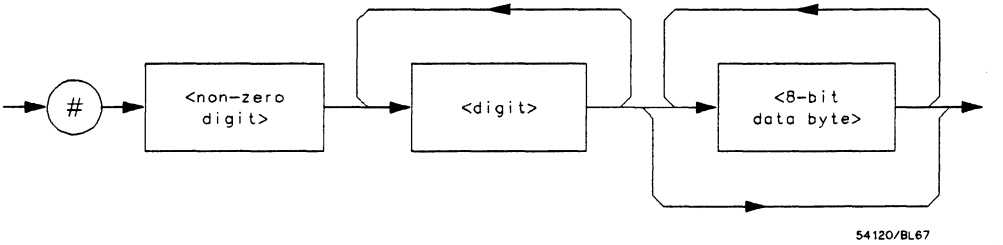
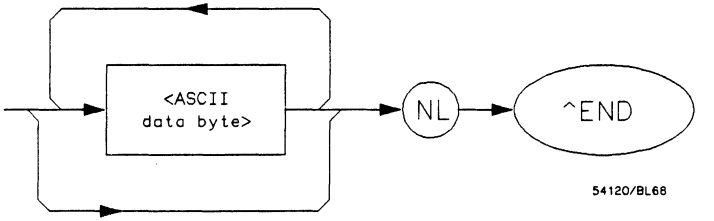


Figure A-26. < definite length arbitrary block response data >



Where <ASCII data byte> represents any ASCII-encoded data byte except <NL> (0A, 10 decimal).

Notes

1. The END message provides an unambiguous termination to an element that contains arbitrary ASCII characters.

2. The IEEE 488.1 END message serves the dual function of terminating this element as well as terminating the <RESPONSE MESSAGE>. It is only sent once with the last byte of the indefinite block data. The NL is present for consistency with the <RESPONSE MESSAGE TERMINATOR>. Indefinite block data format is not supported in the HP 1652B/1653B.

Figure A-27. < arbitrary ASCII response data >

< response data separator >. A comma separates multiple pieces of response data within a single < response message unit >.

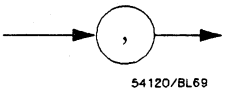


Figure A-28. < response data separator >

< response header separator >. A space (ASCII decimal 32) delimits the response header, if returned, from the first or only piece of data.

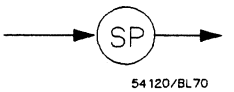


Figure A-29. < response header separator >

< response message unit separator >. A semicolon delimits the < response message unit > s if multiple responses are returned.

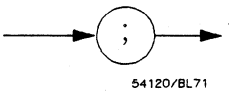


Figure A-30. < response message unit separator >

< response message terminator >. A < response message terminator > (NL) terminates a complete < response message >. It should be read from the instrument along with the response itself.

Common Commands

IEEE 488.2 defines a set of common commands. These commands perform functions which are common to any type of instrument. They can therefore be implemented in a standard way across a wide variety of instrumentation. All the common commands of IEEE 488.2 begin with an asterisk. There is one key difference between the IEEE 488.2 common commands and the rest of the commands found in this instrument. The IEEE 488.2 common commands do not affect the parser's position within the command tree. More information about the command tree and tree traversal can be found in the Programming and Documentation Conventions chapter.

Table A-3. HP 1652B/53B's Common Commands

Command	Command Name
*CLS	Clear Status Command
*ESE	Event Status Enable Command
*ESE?	Event Status Enable Query
*ESR?	Event Status Register Query
*IDN?	Identification Query
*OPC	Operation Complete Command
*OPC?	Operation Complete Query
*RST	Reset (not implemented on HP 1652B/1653B)
*SRE	Service Request Enable Command
*SRE?	Service Request Enable Query
*STB?	Read Status Byte Query
*WAI	Wait-to-Continue Command

Status Reporting B

Introduction

The status reporting feature available over the bus is the serial poll. IEEE 488.2 defines data structures, commands, and common bit definitions. There are also instrument defined structures and bits.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled via the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The "*CLS" command clears all event registers and all queues except the output queue. If "*CLS" is sent immediately following a < program message terminator > , the output queue will also be cleared.

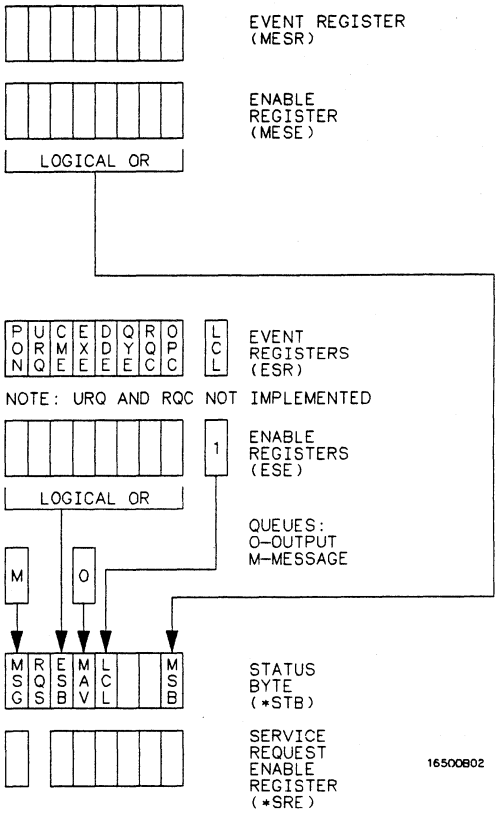


Figure B-1. Status Byte Structures and Concepts

Event Status Register

The Event Status Register is a 488.2 defined register. The bits in this register are "latched." That is, once an event happens which sets a bit, that bit will only be cleared if the register is read.

Service Request Enable Register

The Service Request Enable Register is an 8-bit register. Each bit enables the corresponding bit in the status byte to cause a service request. The sixth bit does not logically exist and is always returned as a zero. To read and write to this register use the *SRE? and *SRE commands.

Bit Definitions

The following mnemonics are used in figure B-1 and in the "Common Commands" chapter:

MAV - message available. Indicates whether there is a response in the output queue.

ESB - event status bit. Indicates if any of the conditions in the Standard Event Status Register are set and enabled.

MSS - master summary status. Indicates whether the device has a reason for requesting service. This bit is returned for the *STB? query.

RQS - request service. Indicates if the device is requesting service. This bit is returned during a serial poll. RQS will be set to 0 after being read via a serial poll (MSS is not reset by *STB?).

MSG - message. Indicates whether there is a message in the message queue (Not implemented in the HP 1652B/1653B).

PON - power on. Indicates power has been turned on.

URQ - user request. Always 0 on the HP 1652B/1653B.

CME - command error. Indicates whether the parser detected an error.

Note 

The error numbers and/or strings for CME, EXE, DDE, and QYE can be read from a device defined queue (which is not part of 488.2) with the query :SYSTEM:ERROR?.

EXE - execution error. Indicates whether a parameter was out of range, or inconsistent with current settings.

DDE - device specific error. Indicates whether the device was unable to complete an operation for device dependent reasons.

QYE - query error. Indicates whether the protocol for queries has been violated.

RQC - request control. Always 0 on the HP 1652B/1653B.

OPC - operation complete. Indicates whether the device has completed all pending operations. OPC is controlled by the *OPC common command. Because this command can appear after any other command, it serves as a general purpose operation complete message generator.

LCL - remote to local. Indicates whether a remote to local transition has occurred.

MSB - module summary bit. Indicates that an enable event in one of the modules Status registers has occurred.

Key Features

A few of the most important features of Status Reporting are listed in the following paragraphs.

Operation Complete. The IEEE 488.2 structure provides one technique which can be used to find out if any operation is finished. The *OPC command, when sent to the instrument after the operation of interest, will set the OPC bit in the Standard Event Status Register. If the OPC bit and the RQS bit have been enabled a service request will be generated. The commands which affect the OPC bit are the overlapped commands.

```
OUTPUT XXX;"*SRE 32 ; *ESE 1"  !enables an OPC service request
```

Status Byte. The Status Byte contains the basic status information which is sent over the bus in a serial poll. If the device is requesting service (RQS set), and the controller serial polls the device, the RQS bit is cleared. The MSS (Master Summary Status) bit (read with *STB?) and other bits of the Status Byte are not be cleared by reading them. Only the RQS bit is cleared when read.

The Status Byte is cleared with the *CLS common command.

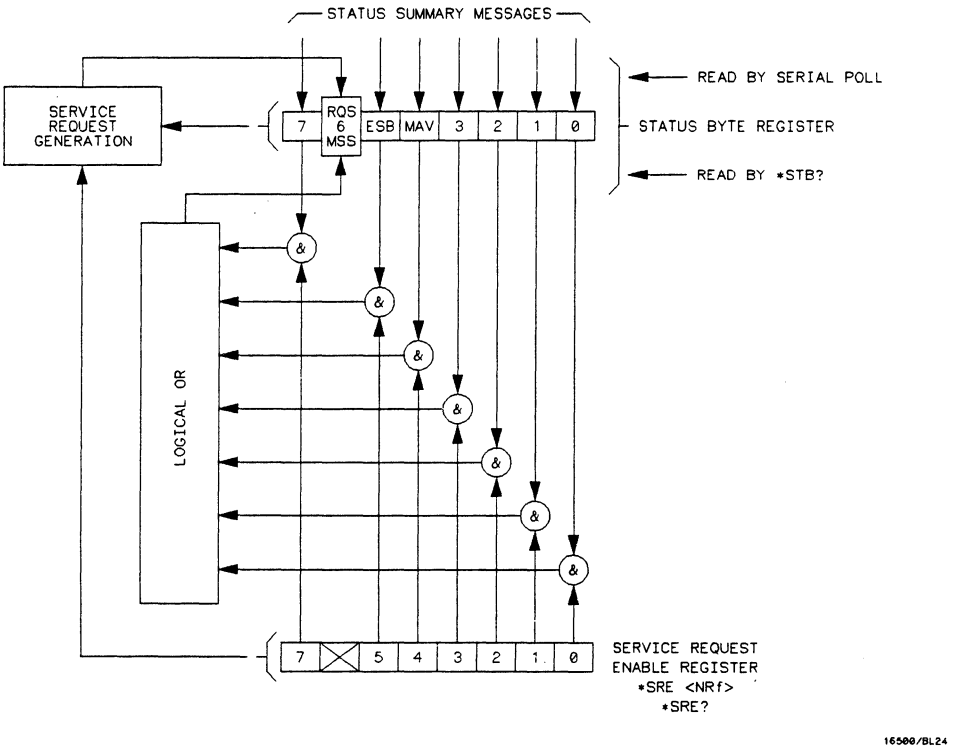


Figure B-2. Service Request Enabling

Serial Poll

The HP 1652B/1653B supports the IEEE 488.1 serial poll feature. When a serial poll of the instrument is requested, the RQS bit is returned on bit 6 of the status byte.

Using Serial Poll (HP-IB)

This example will show how to use the service request by conducting a serial poll of all instruments on the HP-IB bus. In this example, assume that there are two instruments on the bus; a Logic Analyzer at address 7 and a printer at address 1.

The program command for serial poll using HP BASIC 4.0 is Stat = SPOLL(707). The address 707 is the address of the oscilloscope in the this example. The command for checking the printer is Stat = SPOLL(701) because the address of that instrument is 01 on bus address 7. This command reads the contents of the HP-IB Status Register into the variable called Stat. At that time bit 6 of the variable Stat can be tested to see if it is set (bit 6 = 1).

The serial poll operation can be conducted in the following manner:

1. Enable interrupts on the bus. This allows the controller to "see" the SRQ line.

2. Disable interrupts on the bus.

3. If the SRQ line is high (some instrument is requesting service) then check the instrument at address 1 to see if bit 6 of its status register is high.

4. To check whether bit 6 of an instruments status register is high, use the following Basic statement:

```
IF BIT (Stat, 6) THEN
```

5. If bit 6 of the instrument at address 1 is not high, then check the instrument at address 7 to see if bit 6 of its status register is high.

6. As soon as the instrument with status bit 6 high is found check the rest of the status bits to determine what is required.

The SPOLL(707) command causes much more to happen on the bus than simply reading the register. This command clears the bus automatically, addresses the talker and listener, sends SPE (serial poll enable) and SPD (serial poll disable) bus commands, and reads the data. For more information about serial poll, refer to your controller manual, and programming language reference manuals.

After the serial poll is completed, the RQS bit in the HP 1652B/1653B Status Byte Register will be reset if it was set. Once a bit in the Status Byte Register is set, it will remain set until the status is cleared with a *CLS command, or the instrument is reset.

Error Messages C

This section covers the error messages that relate to the HP 1652B/53B Logic Analyzers.

Device Dependent Errors

200 Label not found

201 Pattern string invalid

202 Qualifier invalid

203 Data not available

300 RS-232C error

Command Errors

-100 Command error (unknown command)(generic error)

-101 Invalid character received

-110 Command header error

-111 Header delimiter error

-120 Numeric argument error

-121 Wrong data type (numeric expected)

-123 Numeric overflow

-129 Missing numeric argument

-130 Non numeric argument error (character,string, or block)

-131 Wrong data type (character expected)

-132 Wrong data type (string expected)

-133 Wrong data type (block type #D required)

-134 Data overflow (string or block too long)

-139 Missing non numeric argument

-142 Too many arguments

-143 Argument delimiter error

-144 Invalid message unit delimiter

Execution Errors

-200 No Can Do (generic execution error)

-201 Not executable in Local Mode

-202 Settings lost due to return-to-local or power on

-203 Trigger ignored

-211 Legal command, but settings conflict

-212 Argument out of range

-221 Busy doing something else

-222 Insufficient capability or configuration

-232 Output buffer full or overflow

-240 Mass Memory error (generic)

-241 Mass storage device not present

-242 No media

-243 Bad media

-244 Media full

-245 Directory full

-246 File name not found

-247 Duplicate file name

-248 Media protected

Internal Errors

-300 Device Failure (generic hardware error)

-301 Interrupt fault

-302 System Error

-303 Time out

-310 RAM error

-311 RAM failure (hardware error)

-312 RAM data loss (software error)

-313 Calibration data loss

-320 ROM error

-321 ROM checksum

-322 Hardware and Firmware incompatible

-330 Power on test failed

-340 Self Test failed

-350 Too Many Errors (Error queue overflow)

Query Errors

-400 Query Error (generic)

-410 Query INTERRUPTED

-420 Query UNTERMINATED

-421 Query received. Indefinite block response in progress

-422 Addressed to Talk, Nothing to Say

-430 Query DEADLOCKED

Index

*CLS command 5-3
*ESE command 5-4
*ESR command 5-6
*IDN command 5-8
*OPC command 5-9
*RST command 5-10
*SRE command 5-11
*STB command 5-13
*WAI command 5-15
... 4-3
32767 4-2
9.9E+37 4-2
::= 4-3
4-3
[ ] 4-3
{ } 4-3
| 4-3

A

ACCumulate command/query 14-4, 15-4, 19-6
Acquisition data 6-11
Addressed talk/listen mode 2-1
ALL 27-5
AMODe command/query 18-4
Analyzer 1 Data Information 6-9
Analyzer 2 Data Information 6-11
Angular brackets 4-3
Arguments 1-4
ARM command/query 10-4, 21-4
ARMBnc command 6-4
ASCII Format 26-5
ASSign command/query 10-5
AUToload command/query 7-4
AUToscale 21-5
AUToscale command 10-6
Average Mode 24-2, 26-3

B

BASE command 20-4
Bases 1-8
BASIC 1-2
Baud rate 3-5
Bit definitions B-3
Block data 1-3, 1-16, 6-6
Block length specifier 6-6
Block length specifier 6-7, 6-37
Braces 4-3
BRANch command/query 12-5 - 12-7
BYTE Format 26-4

C

Cable
RS-232C 3-2
CATalog query 7-5
chart display 15-1
Clear To Send (CTS) 3-4
CLOCk command/query 11-4
CMASk command/query 16-4
CME B-3
COLumn command/query 8-3, 13-6 - 13-7

Combining commands 1-5
Comma 1-7
Command 1-3, 1-13
*CLS 5-3
*ESE 5-4
*OPC 5-9
*RST 5-10
*SRE 5-11
*WAI 5-15
ACCumulate 14-4, 15-4, 19-6
AMODe 18-4
ARM 10-4, 21-4
ARMBnc 6-4
ASSign 10-5
AUToload 7-4
AUToscale 10-6, 21-5
BASE 20-4
BRANch 12-5
CLOCk 11-4
CMASk 16-4
COLumn 8-3, 13-6
COMPare 16-3
CONFig 7-9, 7-14
COPY 7-6, 16-5
COUNt 24-4
COUPling 22-4
CPERiod 11-5
DATA 6-5, 16-6
DELay 14-5, 19-7, 25-3
DOWNload 7-7
DSP 6-20
DURation 18-5
EDGE 18-6
FIND 12-8
FORMat 26-10
GLITch 18-8
HAXis 15-5
HEADer 1-12, 6-22
IASSembler 7-10
INITialize 7-8
INSert 14-6, 19-8
Command (continued)
KEY 6-23
LABel 11-6, 17-3
LEVel 23-4
LINE 8-5, 13-9
LOAD:CONFig 7-9
LOAD:IASSembler 7-10
LOCKout 3-7, 6-26
LONGform 1-12, 6-27
MACHine 10-3
MASTer 11-8
MENU 6-28
MESE 6-29
MMODe 13-10, 19-9
MODE 23-5, 25-4
NAME 10-7
OCONdition 19-10
OFFSet 22-5
OPATtern 13-11, 19-11
OSEarch 13-13, 19-13
OTAG 13-15
OTIMe 9-5, 19-14
PACK 7-11
PATTern 18-9, 20-5
PREstore 12-10
PRINt 6-34
PROBe 22-7
PURGe 7-12
RANGe 12-12, 14-7, 16-9, 19-15, 20-6, 22-8, 25-6
RECord 26-13
REMove 11-9, 14-8, 17-5, 19-16, 20-7
REName 7-13
RESTart 12-14
RMODe 6-35
Run Control 6-1
RUNTil 13-16, 16-10, 19-17
SCHart 15-3
SEQuence 12-16
SETup 6-36
SFORmat 11-3
SLAVe 11-10

Command (continued)
SLISt 13-5
SLOPe 23-6
SMODe 21-6
SOURce 23-7, 26-14, 27-14
STARt 6-38
STOP 6-39
STORe 12-17
STORe:CONFig 7-14
STRace 12-4
SWAVeform 14-3
SYMBol 20-3
SYStem:DATA 6-5
SYStem:SETup 6-36
TAG 12-19
TERM 12-21
TFORmat 17-2
THReshold 11-11, 17-6
TTRace 18-3
TWAVeform 19-5
TYPE 10-8, 24-5
VAXis 15-7
WIDTh 20-8
WLISt 9-2
XCONdition 19-24
XPATtern 13-23, 19-26
XSEarch 13-25, 19-28
XTAG 13-27
XTIMe 9-6, 19-29
Command errors C-2
Command mode 2-1
Command set organization 4-10
Command structure 1-11
Command tree 4-4
Command types 4-4
Common commands 1-5, 4-4, 5-1, A-27
Communication 1-2
COMPare selector 16-3
COMPare Subsystem 16-1
Complex qualifier 12-7
Compound commands 1-4
CONFig command 7-9, 7-14
Configuration file 1-10 - 1-11
Controller mode 2-1
Controllers 1-2
Conventions 4-3
COPY command 7-6, 16-5
COUNt 24-4
COUNt query 26-8
COUPling 22-4
CPERiod command/query 11-5

D

DATA 6-5, 26-9
command 6-5
State (no tags) 6-12
State (with either time or state tags) 6-12
Timing Glitch 6-14
Transitional Timing 6-15
Data bits 3-5 - 3-6
8-Bit mode 3-6
Data block
Acquisition data 6-11
Analyzer 1 data 6-9
Analyzer 2 data 6-11
Data preamble 6-8
Section data 6-8
Section header 6-8
Data Carrier Detect (DCD) 3-4
DATA command/query 6-5 - 6-19, 16-6 - 16-7
Data Communications Equipment 3-1
Data mode 2-1
Data preamble 6-8
DATA query 13-8
Data Set Ready (DSR) 3-4
Data Terminal Equipment 3-1
Data Terminal Ready (DTR) 3-3
DCE 3-1
DCL 2-3
DDE B-4

Definite-length block response data 1-16
Definitions 4-3
DELay 25-3
DELay command/query 14-5, 19-7
Device address 1-3
  HP-IB 2-2
  RS-232C 3-6
Device clear 2-3
Device dependent errors C-1
DLISt
  Command 8-2
DLISt selector 8-2
DLISt Subsystem 8-1
Documentation conventions 4-3
DOWNload command 7-7
DSP command 6-20
DTE 3-1
Duplicate keywords 1-5
DURation command/query 18-5

E

EDGE command/query 18-6 - 18-7
EDGE Trigger Mode 23-1
Ellipsis 4-3
Embedded strings 1-2 - 1-3
Enter statement 1-2
Error messages C-1
ERRor query 6-21
ESB B-3
Event Status Register B-3
EXE B-4
Execution errors C-3
Exponents 1-8
Extended interface 3-3

F

FALLtime 27-6
FIND command/query 12-8 - 12-9
FIND query 16-8
FORMat 26-10
Fractional values 1-8
FREQuency 27-7

G

GET 2-3
GLITch command/query 18-8
Glitch Timing Data 6-14
Group execute trigger 2-3

H

HAXis command/query 15-5 - 15-6
HEADer command 1-12
HEADer command/query 6-22
Headers 1-3 - 1-4, 1-7
Host language 1-3
HP-IB 2-1, B-6
HP-IB address 2-1
HP-IB device address 2-2
HP-IB interface 2-1
HP-IB interface code 2-2
HP-IB interface functions 2-1

I

IASSembler command 7-10
IEEE 488.1 2-1, A-1

IEEE 488.1 bus commands 2-3
IEEE 488.2 A-1
IEEE 488.2 Standard 1-1
IFC 2-3
Infinity 4-2
Initialization 1-10
INITialize command 7-8
Input buffer A-2
INSert command 14-6, 19-8
Instruction headers 1-3
Instruction parameters 1-4
Instruction syntax 1-2
Instruction terminator 1-9
Instructions 1-3
Instrument address 2-2
Interface capabilities 2-1
  RS-232C 3-5
Interface clear 2-3
Interface code
  HP-IB 2-2
Interface select code
  RS-232C 3-6
Internal errors C-4

K

KEY command/query 6-23
Keyword data 1-8
Keywords 4-1

L

LABel command/query 11-6 - 11-7, 17-3 - 17-4
LCL B-4
LER query 6-25
LEVel 23-4
LINE command/query 8-5, 13-9
Linefeed 1-9, 4-3
Listening syntax A-8
LOAD:CONFig command 7-9
LOAD:IASSembler command 7-10
Local 2-2
Local lockout 2-2
LOCKout command 3-7
LOCKout command/query 6-26
Longform 1-7
LONGform command 1-12
LONGform command/query 6-27
Lowercase 1-7

M

Machine selector 10-3
MACHine Subsystem 10-1
MASTer command/query 11-8
MAV B-3
MENU command/query 6-28
MESE command/query 6-29
MESR query 6-31 - 6-32
MMEMory subsystem 7-1
MMODe command/query 13-10, 19-9
Mnemonics 1-8, 4-1
MODE 23-5, 25-4 - 25-5
Module Level Commands 21-1
MSB B-4
MSG B-3
MSS B-3
Multiple numeric variables 1-17
Multiple program commands 1-9
Multiple queries 1-17
Multiple subsystems 1-9

N

NAME command/query 10-7
New Line character 1-9

NL 1-9, 4-3
Normal Mode 24-2, 26-3
Notation conventions 4-3
Numeric base 1-15
Numeric bases 1-8
Numeric data 1-8
Numeric variables 1-15
NWIDth 27-8

O

OCONdition command/query 19-10
OFFSet 22-5 - 22-6
OPATtern command/query 13-11 - 13-12, 19-11 - 19-12
OPC B-4
Operation Complete B-4
OR notation 4-3
oscilloscope 21-1
Oscilloscope Subsystem commands 21-1
OSEarch command/query 13-13, 19-13
OSTate 13-14
OSTate query 9-3
OTAG command/query 13-15
OTIMe command/query 9-5, 19-14
Output buffer 1-6
Output command 1-3
Output queue A-2
OUTPUT statement 1-2
Overlapped command 5-9, 5-15, 6-38 - 6-39
Overlapped commands 4-2
OVERshoot 27-9

P

PACK command 7-11
Parameter syntax rules 1-7
Parameters 1-4
Parity 3-5
Parse tree A-7
Parser A-2
PATTern command 20-5
PATTern command/query 18-9 - 18-10
PATTern Trigger Mode 23-1
PERiod 27-10
POINts query 26-11
PON B-3
PPOWer query 6-33
PREamble 26-12
Preamble description 6-8
PREShoot 27-11
PREstore command/query 12-10 - 12-11
PRINt command 6-34
Printer mode 2-1
PROBe 22-7
Program data A-14
Program examples 4-11
Program message A-9
Program message syntax 1-2
Program message terminator 1-9
Program syntax 1-2
Programming conventions 4-3
Protocol 3-5, A-3
  None 3-5
  XON/XOFF 3-5
Protocol exceptions A-4
Protocols A-2
PURGe command 7-12
PWIDth 27-12

Q

Query 1-3, 1-6, 1-13
  *ESE 5-4
  *ESR 5-6
  *IDN 5-8
  *OPC 5-9
  *SRE 5-11

Query (continued)
*STB 5-13
ACCumulate 14-4, 15-4, 19-6
ALL 27-5
AMODe 18-4
ARM 10-4, 21-4
ARMBnc 6-4
ASSign 10-5
AUToload 7-4
BRANch 12-5
CATalog 7-5
CLOCk 11-4
CMASk 16-4
COLumn 8-3, 13-6
COUNt 24-4, 26-8
COUPling 22-4
CPERiod 11-5
DATA 6-5, 13-8, 16-6, 26-9
DELay 14-5, 19-7, 25-3
DURation 18-5
EDGE 18-6
ERRor 6-21
FALLtime 27-6
FIND 12-8, 16-8
FORMat 26-10
FREQuency 27-7
GLITch 18-8
HAXis 15-5
HEADer 6-22
KEY 6-23
LABel 11-6, 17-3
LER 6-25
LEVel 23-4
LINE 8-5, 13-9
LOCKout 6-26
LONGform 6-27
MASTer 11-8
MENU 6-28
MESE 6-29
MESR 6-31
MMODe 13-10, 19-9

Query (continued)
MODE 23-5, 25-4
NAME 10-7
NWIDth 27-8
OCONdition 19-10
OFFSet 22-5
OPATtern 13-11, 19-11
OSEarch 13-13, 19-13
OSTate 9-3, 13-14
OTAG 13-15
OTIMe 9-5, 19-14
OVERshoot 27-9
PATTern 18-9
PERiod 27-10
POINts 26-11
PPOWer 6-33
PREamble 26-12
PREShoot 27-11
PROBe 22-7
PWIDth 27-12
RANGe 12-12, 14-7, 16-9, 19-15, 22-8, 25-6
RECord 26-13
RESTart 12-14
RISetime 27-13
RMODe 6-35
RUNTil 13-16, 16-10, 19-17
SEQuence 12-16
SETup 6-36
SLAVe 11-10
SLOPe 23-6
SMODe 21-6
SOURce 23-7, 26-14, 27-14
SPERiod 19-19
STORe 12-17
SYSTem:DATA 6-5
SYStem:SETup 6-36
TAG 12-19
TAVerage 13-18, 19-20
TERM 12-21
THReshold 11-11, 17-6
TMAXimum 13-19, 19-21

Query (continued)
TMINimum 13-20, 19-22
TYPE 10-8, 24-5, 26-15
UPLoad 7-15
VALid 26-16
VAMPlitude 27-15
VAXis 15-7
VBASe 27-16
VMAX 27-17
VMIN 27-18
VPP 27-19
VRUNs 13-21, 19-23
VTOP 27-20
XCONdition 19-24
XINCrement 26-17
XORigin 26-18
XOTag 13-22
XOTime 19-25
XPATtern 13-23, 19-26
XREFerence 26-19
XSEarch 13-25, 19-28
XSTate 9-4, 13-26
XTAG 13-27
XTIMe 9-6, 19-29
YINCrement 26-20
YORigin 26-21
YREFerence 26-22
Query errors C-5
Query responses 1-11, 4-2
Question mark 1-6
QYE B-4

R

RANGe 22-8, 25-6
RANGe command 20-6
RANGe command/query 12-12 - 12-13, 14-7, 16-9, 19-15
Receive Data (RD) 3-2 - 3-3
record 26-13
waveform 26-3
Remote 2-2
Remote enable 2-3
REMove command 11-9, 14-8, 17-5, 19-16, 20-7
REN 2-3
REName command 7-13
Request To Send (RTS) 3-4
Response data 1-16
Response message A-21
Responses 1-12
RESTart command/query 12-14 - 12-15
RISetime 27-13
RMODe command/query 6-35
Root 4-4
RQC B-4
RQS B-3
RS-232C 3-1, 3-6, A-1
Run Control Commands 6-1
RUNTil command/query 13-16 - 13-17, 16-10 - 16-11, 19-17 - 19-18

S

SCHart selector 15-3
SCHart Subsystem 15-1
SCOPe Subsystem 21-1
SDC 2-3
Section data 6-8
Section data format 6-6
Section header 6-8
Selected device clear 2-3
Separator A-18
SEQuence command/query 12-16
Sequential commands 4-2
Serial poll B-6
Service Request Enable Register B-3
SETup 6-36
SETup command/query 6-36 - 6-37
SFORmat selector 11-3

SFORmat Subsystem 11-1
Shortform 1-7
Simple commands 1-4
SLAVe command/query 11-10
SLISt selector 13-5
SLISt Subsystem 13-1
SLOPe 23-6
SMODe command 21-6
SMODe query 21-6
SOURce 23-7, 26-14, 27-14
Spaces 1-4
SPERiod query 19-19
Square brackets 4-3
STARt command 6-38
State data
  with either time or state tags 6-12
  without tags 6-12
Status 1-17, 5-2, B-1
Status byte B-5
Status registers 1-17
Status reporting B-1
Stop bits 3-5
STOP command 6-39
STORe command/query 12-17 - 12-18
STORe:CONFig command 7-14
STRace selector 12-4
STRace Subsystem 12-1
String data 1-8
String variables 1-14
Subsystem
  ACQuire 24-1
  CHANnel 22-1
  COMPare 16-1
  DLIST 8-1
  MACHine 10-1
  MEASure 27-1
  MMEMory 7-1
  SCHart 15-1
  SCOPe 21-1
  SFORmat 11-1
  SLISt 13-1
Subsystem (continued)
  STRace 12-1
  SWAVeform 14-1
  SYMBol 20-1
  TFORmat 17-1
  TIMebase 25-1
  TRIGger 23-1
  TTRace 18-1
  TWAVeform 19-1
  WAVeform 26-1
  WLISt 9-1
Subsystem commands 4-4
Suffix multiplier A-16
Suffix units A-16
SWAVeform selector 14-3
SWAVeform Subsystem 14-1
SYMBol selector 20-3
SYMBol Subsystem 20-1
Syntax A-8
Syntax diagram
  ACQuire Subsystem 24-1
  CHANnel Subsystem 22-2
  Common commands 5-2
  COMPare Subsystem 16-2
  DLISt Subsystem 8-1
  MACHine Subsystem 10-2
  MEASure Subsystem 27-3
  MMEMory subsystem 7-2 - 7-3
  SCHart Subsystem 15-2
  SCOPe Subsystem 21-1
  SFORmat Subsystem 11-1
  SLISt Subsystem 13-2
  STRace Subsystem 12-1
  SWAVeform Subsystem 14-2
  SYMBol Subsystem 20-2
  System commands 6-3
  TFORmat Subsystem 17-1
  TIMebase Subsystem 25-1
  TRIGger Subsystem 23-2
  TTRace Subsystem 18-2
  TWAVeform Subsystem 19-2

Syntax diagram (continued)
WAVeform Subsystem 26-2
WLISt Subsystem 9-1
Syntax diagrams 4-2
IEEE 488.2 A-5
System commands 4-4, 6-1

T

TAG command/query 12-19 - 12-20
Talk only mode 2-1
Talking syntax A-21
TAVerage query 13-18, 19-20
TERM command/query 12-21 - 12-22
Terminator 1-9, A-26
TFORmat selector 17-2
TFORmat Subsystem 17-1
Three-wire Interface 3-2
Threshold command/query 11-11, 17-6
Timing Glitch Data 6-14
TMAXimum query 13-19, 19-21
TMINimum query 13-20, 19-22
Trailing dots 4-3
Transitional Timing Data 6-15
Transmit Data (TD) 3-2 - 3-3
Truncation rule 4-1
TTRace selector 18-3
TTRace Subsystem 18-1
TWAVeform selector 19-5
TWAVeform Subsystem 19-1
TYPE 24-5
TYPE command/query 10-8
TYPE query 26-15

U

Units 1-8
UPLoad query 7-15
Uppercase 1-7
URQ B-3

V

VALid 26-16
VAMPlitude 27-15
VAXis command/query 15-7
VBASe 27-16
VMAX 27-17
VMIN 27-18
VPP 27-19
VRUNs query 13-21, 19-23
VTOP 27-20

W

waveform
record 26-3
White space 1-4
WIDTh command 20-8
WLISt selector 9-2
WLISt Subsystem 9-1
WORD Format 26-5

X

XCONdition command/query 19-24
XINCrement query 26-17
XORigin query 26-18
XOTag query 13-22
XOTime query 19-25
XPATtern command/query 13-23 - 13-24, 19-26 - 19-27
XREFerence query 26-19
XSEarch command/query 13-25, 19-28
XSTate query 9-4, 13-26

XTAG command/query 13-27
XTIMe command/query 9-6, 19-29
XXX 4-3, 4-5
XXX (meaning of) 1-3

Y

YINCrement query 26-20
YORigin query 26-21
YREFerence query 26-22

Your Comments Please

HP 1650B/1651B Programming

Your comments assist us in meeting your needs better. Please complete this questionnaire and return it to us. Feel free to add any additional comments that you might have. All comments and suggestions become the property of Hewlett-Packard. Omit any questions that you feel would be proprietary.

	Yes	No
1. Were you satisfied with the operation of the instrument over the bus?	[ ]	[ ]

2. What measurements will this instrument be used to make over the bus?

_____

3. What type of controller are you using?_____

4. What programming language are you using?_____

5. What do you like most about programming the instrument?_____

_____

6. What would you like to see changed or improved?_____

_____

7. Which sections of the manual have you used?

[ ] Introductory chapters 1 through 4
[ ] Command List chapters 5 through 27
[ ] Appendix A
[ ] Programming Examples
[ ] Index

8. Please rate the manual on the following:

4 = Excellent 3 = Good 2 = Adequate 1 = Poor

[ ] Breadth and depth of information
[ ] Ability to easily find information
[ ] Ability to understand and apply the information provided in the manual

Please explain:_____

9. What is your experience with programming instruments over the bus?

[ ] No previous experience
[ ] Less than 1 year experience
[ ] More than 1 year's experience on one model
[ ] More than 1 year's experience on several models

Name _____ Company _____
Address _____ Zip Code _____
Phone _____ Instrument Serial # _____

THANK YOU FOR YOUR HELP

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

Your cooperation in completing and returning this form will be greatly appreciated. Thank you.

FOLD HERE

HEWLETT PACKARD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 1303 COLORADO SPRINGS, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD

COLORADO SPRINGS DIVISION
ATTN: PUBLICATIONS DEPT.
P.O. BOX 2197
COLORADO SPRINGS, COLORADO 80901-9959



Printed in U.S.A.